ELSEVIER

Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Logic and Algebraic Programming 62 (2005) 191–245

THE JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING
www.elsevier.com/locate/jlap

Hybrid process algebra

P.J.L. Cuijpers*, M.A. Reniers

Eindhoven University of Technology (TU/e), Den Dolech 2, Eindhoven 5600 MB, The Netherlands

Received 13 October 2003; accepted 17 February 2004

Abstract

We develop an algebraic theory, called hybrid process algebra (HyPA), for the description and analysis of hybrid systems. HyPA is an extension of the process algebra ACP, with the disrupt operator from LOTOS and with flow clauses and re-initialization clauses for the description of continuous behavior and discontinuities. The semantics of HyPA is defined by means of deduction rules that associate a hybrid transition system with each process term. A large set of axioms is presented for a notion of bisimilarity. HyPA may be regarded as an algebraic approach to hybrid automata, although the specific semantics of re-initialization clauses makes HyPA a little more expressive.

Keywords: Hybrid systems; Process algebra; Flows; Discrete events; Hybrid interaction; Discontinuities

1. Introduction

1.1. Hybrid systems

The theory of hybrid systems studies the combination of continuous/physical and discrete/computational behavior. When computational software is combined with mechanical and electrical components, or is interacting with, for example, chemical processes, a hybrid system arises in which the interaction between the continuous behavior of the components, and the discrete behavior of the software is important.

In current practice, often the discrete part of a hybrid system is described and analyzed using methods from computer science, while the continuous part is handled by control science. The design of the complete system is usually such that interaction between the discrete and continuous part is suppressed to a minimum. Because of this suppressed interaction, analysis is possible to some extent, but it limits the design options. In the field of hybrid systems theory, researchers attempt to extend the possibilities for interaction. The goal of this paper is to develop an algebraic theory, called hybrid process algebra

* Corresponding author.
E-mail addresses: p.j.l.cuijpers@tue.nl (P.J.L. Cuijpers), m.a.reniers@tue.nl (M.A. Reniers).

1567-8326/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2004.02.001

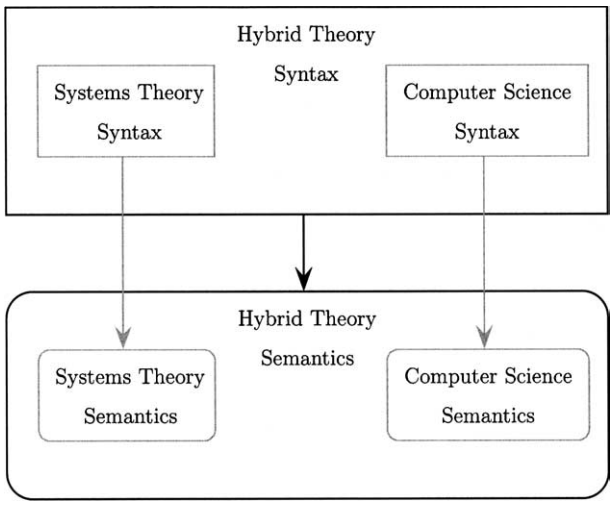

Fig. 1. Developing hybrid theory.

(HyPA), to support these attempts. Our hopes are that hybrid process algebra can serve as a mathematical basis for improvement of the design strategies of hybrid systems, and the possibilities to analyze them.

In Fig. 1, a graphical representation is given of the general aim of our efforts. The figure shows our desire, that a hybrid theory is, in a sense, a conservative extension of computer science and systems theory. More precisely, a model from systems theory or computer science, should be expressible, and preferably look the same, in the hybrid theory, and theorems from systems theory and computer science should be transferable to the hybrid theory (when restricted to models from the original field of course). What the figure does not show, is that this conservativity is not the only goal. In that case, a simple union of the theories would be sufficient. We also desire a certain interaction between the theories, reflecting the interaction between software and physics described before. This goal is harder to formalize, but in the remainder of this introduction we hope to give some feeling for it, using examples of deficiencies (in our view) in existing hybrid formalisms, and indicating how we intend to improve on those.

1.2. Algebraic reasoning

In systems theory, algebraic reasoning is acknowledged by most people, as one of the most powerful tools available for analyzing physical behavior. This behavior is usually described by differential equations and inclusions, which model the rate of change of the value of certain continuous variables, and algebraic equations or inequalities modeling constraints. When certain abstractions are made on physical systems [1], also discontinuous behavior is sometimes relevant, which is often described using difference equations to model changes and algebraic inequalities to model constraints. In this paper, we use a slight generalization of these modeling formalisms, in the form of flow clauses for continuous behavior, and re-initialization clauses for discontinuous behavior. This generalization was inspired by the work of [2].

In computer science, the usefulness of algebra is still a topic of much debate, but nevertheless there are interesting examples of applications of process algebra (see for example

[3] for a list of references to protocol verifications, [4,5] for a start in the description and analysis of other industrial size problems, like the design of a controller for a coating system and a turntable system, and [6] for the description and analysis of railway interlocking specifications). In process algebra, the discrete actions that a system may perform are often considered atomic elements of the algebraic description language. These 'atomic actions' can be combined using compositional operators describing choice between behaviors, sequential execution of behaviors, and concurrent execution of behaviors.

In this paper, we attempt to combine the compositional view on systems that process algebra gives us, with the continuous and discontinuous physical behaviors described by systems theory. To this end, we take the process algebra ACP [7] and extend it with a new atom, describing continuous behavior through the use of flow clauses, and with a new family of unary operators, describing discontinuous behavior through re-initialization clauses, as mentioned before. Also, we import the disrupt operator from LOTOS [8], since it turns out to model the sequential composition of flow clauses well. The choice for ACP is rather arbitrary, and we expect that the methods described in this paper can be easily extended to other process algebras.

So far, the only algebraic approaches that we know of regarding hybrid systems, are described in [9–11] (hybrid χ), [12,13] (hybrid versions of ACP), [14] (hybrid CSP) and [15] (ϕ-calculus). In the remainder of this introduction, we explain the deficiencies that these methods have, in our opinion, in describing hybrid interaction. We should note, that within other hybrid formalisms like hybrid automata [16,17], hybrid Petri nets [18–22] and hybrid action systems [23], the use of algebraic reasoning on differential equations for analysis purposes, is not uncommon. It is the process algebraic reasoning that is underexposed. For a translation of hybrid automata into the process algebras CSP, timed μCRL, and hybrid χ, see [24], [25,26], and [10], respectively.

In the hybrid theory that has been developed by system theorists (see for example [2,27–31]) algebraic reasoning is possible, but none of these theories support reasoning about non-determinism. All of these theories have a trace semantics, and cannot distinguish between processes that only differ in their non-deterministic choices. Since we would like a conservative extension of process algebra, we would also like to be able to distinguish systems up to the notion of bisimilarity, and therefore, we consider the system theoretic formalisms as non-conserv--break ative with respect to computer science. We should note here, that first investigations into what the notion of bisimilarity means for continuous systems, can be found in [32,33].

In Section 3, we prove formally that HyPA is a conservative extension of the process algebra ACP, and by construction of the semantics, it is immediately clear that it is a conservative extension of differential inclusions and difference equations.

1.3. Flows and re-initializations

Before we discuss our views on hybrid interaction and on discontinuities, which are crucial to some of the choices made in the development of HyPA, we have to explain the concepts of flow and re-initialization, and illustrate the way they are described traditionally, and in this paper.

As mentioned before, continuous physical behavior is often modeled through differential equations and algebraic inequalities, while discontinuous physical behavior is modeled in a similar way through difference equations and algebraic inequalities. As an example of a differential equation, take $\dot{x} = f(x, u)$, in which x and u are variables ranging over the real numbers, and f is a real-valued function. This equation models that the value of

x changes continuously through time (indicated by the dot in $\dot{x}$) with a rate defined by $f(x, u)$, i.e. by a function of the current value of x and u. Alternatively, if there is a choice of rates of change, one may write $\dot{x} \in F(x, u)$, in which F is a set-valued function over the reals. Also, an inequality $x \leqslant f(x, y)$ may denote that x is constrained in its value (not its rate of change) for some reason. As an example of a difference equation, $x^+ = f(x^-, u^-)$ denotes that the value of x is reassigned to $f(x^-, u^-)$, based on the previous values of x and u. This notation is for example used in [2].

More generally, differential equations and algebraic inequalities form predicates on the flow of variables, where a flow is simply a function of time to valuations of variables. Difference equations are predicates about the re-initialization (or discontinuity) of variables. In systems theory, several different formalisms are used for the description of continuous and discontinuous behavior, and often the modeling or analysis question determines which formalism is to be used. For example, integral equations are sometimes easier to use than differential equations, and sometimes even the notion of solution for a differential equation can vary (although not within one model). The consequence for our hybrid approach, is that we have to parameterize our theory in such a way that instantiations of these different formalisms can be chosen at will, by the modeler.

Flow predicates, and their notion of solution, parameterize the modeling of continuous, never terminating, physical behavior, by describing how model variables $\mathcal{V}_\mathrm{m}$ are allowed to change through time. A flow predicate describes a set of flows, where a flow is a (partial) function of time T (some totally ordered set with a least element denoted 0) with a closed-interval domain starting from 0, to the valuations of model variables $\mathcal{V}_\mathrm{m}$. Both the model variables $\mathcal{V}_\mathrm{m}$ (including the domains they range over) and an appropriate notion of time T are problem-specific and should be given by the modeler. The domain $\mathcal{V}(x)$ of a model variable $x \in \mathcal{V}_\mathrm{m}$ is specified by the modeler at the first introduction of the variables. In this paper, the specification of domains is left out since, most of the time, it is obvious from the context. Flow predicates are a core part of the flow clauses of HyPA, that are formally defined in Section 2.1.

Formally, we write $\mathcal{V} = \bigcup_{x \in \mathcal{V}_\mathrm{m}} \mathcal{V}(x)$ for the union of all variable domains, and $\mathit{Val} = \mathcal{V}_\mathrm{m} \to \mathcal{V}$ for the set of variable valuations. The set of all flows with a closed-interval domain starting in 0 is $\mathcal{F} = \{ f \in T \mapsto \mathit{Val} \mid \mathrm{dom}(f) = [0, t] \text{ for some } t \in T \}$. The flows that are described by a flow predicate, are called solutions of that predicate. We consider the set of flow predicates $\mathcal{P}_\mathrm{f}$, the sets $\mathcal{V}_\mathrm{m}$ of model variables and T of time points, and the notion of solution $\models_\mathrm{f} \subseteq \mathcal{F} \times \mathcal{P}_\mathrm{f}$, that defines which flows are considered solutions of a flow predicate, parameters of the theory. This means they can be instantiated by the modeler, depending on the specific modeling or analysis problem. The theory we present in this paper, is largely independent of that choice, except that we assume the existence of a flow predicate $\mathit{false} \in \mathcal{P}_\mathrm{f}$ that satisfies no flow from the set $\mathcal{F}$.

Re-initialization predicates describe a set of re-initializations, which are pairs of valuations representing the values of the model variables prior to and immediately after the re-initialization. Such re-initializations are called solutions of the re-initialization predicate. The set of all re-initializations $\mathit{Val} \times \mathit{Val}$ is denoted $\mathcal{R}$. As before, the set of re-initialization predicates $\mathcal{P}_\mathrm{r}$ and the notion of solution $\models_\mathrm{r} \subseteq \mathcal{R} \times \mathcal{P}_\mathrm{r}$, that defines which re-initializations are considered solutions of a re-initialization predicate, are considered parameters of the theory. We assume the existence of re-initialization predicates $\mathit{true}, \mathit{false} \in \mathcal{P}_\mathrm{r}$ that satisfy any re-initialization, and no re-initialization from the set $\mathcal{R}$, respectively. Re-initialization predicates are a core part of the re-initialization clauses of HyPA, defined in Section 2.1.

Hybrid process algebra, intends to reason about predicates on flows, and about predicates on re-initializations, in general. However, since the use of differential and algebraic

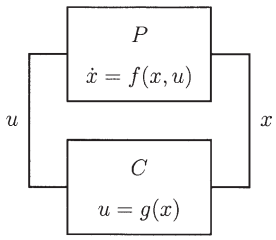

Fig. 2. Continuous control system.

equations is common, we make use of this particular kind of predicates in the examples that we give. In this article, a flow predicate is specified as a differential or algebraic equation on the variables $\mathcal{V}_{\mathrm{m}}$ and their derived[1] versions $\dot{\mathcal{V}}_{\mathrm{m}} = \{\dot{x} \mid x \in \mathcal{V}_{\mathrm{m}}\}$ (with $\dot{x}$ also taking value in $\mathcal{V}(x)$). Typical flow predicates are, for example $\dot{x} = f(x, y)$, and $x \leqslant f(x, y)$. For the description of re-initialization predicates in our examples, we make use of the sets of variables $\mathcal{V}_{\mathrm{m}}^{-} = \{x^{-} \mid x \in \mathcal{V}_{\mathrm{m}}\}$ and $\mathcal{V}_{\mathrm{m}}^{+} = \{x^{+} \mid x \in \mathcal{V}_{\mathrm{m}}\}$, modeling the current and future value of a model variable, respectively. Typical re-initialization predicates are assignments, for example $x^{+} = f(x^{-}, y^{-})$ which, in imperative programming, is usually denoted as $x := f(x, y)$. But, also boolean predicates can be modeled using only the current value of variables, for example $x^{-} \leqslant y^{-}$, which only allows discontinuities if x is smaller than y to start with. If necessary, this can be combined with equations $x^{-} = x^{+}$ and $y^{-} = y^{+}$, enforcing that the values of x and y actually do not change. In Section 2.1, re-initialization clauses are introduced formally in such a way that this enforcement can be done more efficiently. In the remaining parts of this section, the above notations will be used to illustrate our reasons for certain choices in the development of HyPA.

1.4. Hybrid interaction

Many of the hybrid formalisms that we mentioned in Section 1.2, have some problem in the definition of parallel composition. Surprisingly, in most cases, this problem comes to light in a purely continuous case study. Let us consider the following example, depicted in Fig. 2, of a continuous plant P described by the differential equation $\dot{x} = f(x, u)$, and a continuous controller C described by $u = g(x)$. The composition of plant and controller is denoted $P \parallel C$.

The hybrid automata of Henzinger [16], as well as the hybrid process algebras of Vereijken [12] and of Jifeng [14], assume that the continuous behavior of two composed systems is independent. Using these formalisms, the system $P \parallel C$ would not model any interaction between P and C at all, since the only interaction between systems can be through computational actions. The variable x of P would simply be regarded different from the variable x of C. Hence, in our opinion, these formalisms cannot be considered to be a conservative extension of systems theory. At least, they do not support the way in which we would like to think about parallel composition of systems. In the semantics of the tool HyTech [34,35], shared continuous variables do not pose a problem, because a hybrid

[1] We assume derivation is defined for all model variables, but if we want to use a variable x for which this is not the case (for example a computational data structure), then no formal problems arise as long as we do not use the derived variable $\dot{x}$ in our predicates. In such cases, the value of x is assumed constant throughout the flow.

trace semantics is used for Henzinger's hybrid automata, rather than a timed transition system semantics. This formalism is not suitable for us, however, since it is not algebraic, and only supports a restricted class of differential equations.

More surprisingly, it turns out that the parallel composition of the above processes is not defined for the hybrid I/O automaton model of Lynch et al. [17] either, at least not without a few amendments. In the formalism of [17], it is necessary to identify variables as either state variables of a system, or as external variables of the system. These two sets of variables are supposed to be disjoint. The intuition behind this partition is that the state variables model the memory of the system, while the external variables model the communication with other systems. Therefore, in a parallel composition, it is required that two hybrid I/O automata are *compatible*, meaning that the state variables of the one automaton do not intersect with any of the variables of the other automaton. Now, looking at the plant P of Fig. 2, we see that we need to choose x to be a state variable, otherwise information on x is lost between transitions, but it also needs to be an external variable, since we need to communicate its value with the controller C. This contradicts the requirement on hybrid I/O automata that the set of state variables and the set of external variables are disjoint. The problem is not as big as it may seem, since by adding an external variable y, and the equation $y = x$, to the description of P, and changing the description of C to $u = g(y)$, we can declare x to be a state variable, and find that the systems have become compatible. So, although the system in Fig. 2 cannot be modeled as $P \parallel C$ directly in this hybrid I/O automaton model, we can model the modification depicted in Fig. 3 instead.

In [36] it was already noted that the partitioning of the variables of a system into state variables and external variables is not always uniquely determined by the equations that describe the system. Even in our simple control example, it is possible to use the equations $x = y$ and $u = g(x)$, and declare x external and y a state variable. Often, there is no clear physical ground to choose a specific partition. This is one reason why we would like to avoid the partitioning of the set of variables of a system, in our semantics. Another reason, is that in basic textbooks on control theory (for example [37]), one usually starts out with developing controllers for plants of which the state variables are also output variables. It therefore seems, that the intuition behind compatibility, that state variables do not play a role in communication with other systems, does not coincide with the system-theoretic intuition. This is confirmed by the theory discussed in [36], where state variables may also be output variables of a system, while external variables may be inputs or outputs. In this paper, we show that partitioning the model variables as done for hybrid automata, is in fact not necessary, if a slightly different semantical view is taken.

HyPA is developed in close cooperation with the people working on the formal semantics of the language hybrid χ, which is focussed on the simulation of hybrid systems.

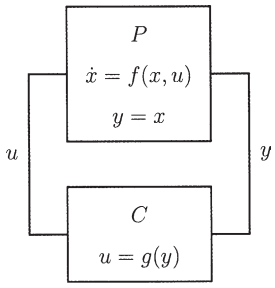

Fig. 3. Compatible continuous control system.

Their operational semantics [11] uses a semantical structure similar to, and based on, the one we have developed for HyPA (discussed in Section 2.2). Also the hybrid process algebra of Bergstra and Middelburg [13] uses a hybrid transition system semantics. In Section 4, we discuss the relation between HyPA, hybrid χ and the process algebra of [13] in more detail. Admittedly, these three languages are very similar, which calls for a more thorough comparison in the near future.

In ϕ-calculus [15], the semantics assumes continuous behavior to be a property of the environment, rather than of the process itself. There, (urgent) environmental actions allow the process to change the rules for continuous behavior, specified by differential equations and invariants, in an interleaving manner. This leads to the consistent update of the differential equations and invariants in the environment. The semantics of ϕ-calculus in such, that the environment can only execute time transitions, if the total set of differential equations that is placed in the environment is autonomous. Since ϕ-calculus only takes the differential equations into account for autonomicity, the environment resulting from $P \| C$ is not considered autonomous. This, ultimately, leads to a deadlocking situation in the process $P \| C$. In ϕ-calculus the processes $\dot{x} = f(x, u) \| u = g(x)$ and $\dot{x} = f(x, g(x)) \| u = g(x)$ are not equivalent. These observations contradict with our intuition on the parallel composition.

In hybrid action systems, the parallel composition of P and C leads to the desired result, ignoring some syntactic differences. However, the parallel composition of two differential equations $\left\lfloor \dot{x} = 1 \right\rfloor \parallel \left\lfloor \dot{x} = 2 \right\rfloor$ results in a process that acts like the differential inclusion $\dot{x} \in \{1, 2\}$. This, again, contradicts with our intuition. We would expect contradicting equations to result in deadlock. Nevertheless, both the 'interleaving' approaches from ϕ-calculus and hybrid action systems, might turn out to be useful in situations where our intuition is flawed, and the theories might be considered complementary to HyPA.

In conclusion, we might state that we aim for an algebraic formalism, in which the parallel composition has a similar intuition as in [17], but without having to require compatibility of the composed systems. To do this, we have worked out the notion of *hybrid transition system*, as a semantical framework, in [38]. This framework, formally defined in Section 2.2, unifies the discrete behavior of computer science and the continuous behavior of system theory in a similar way as the hybrid automata of [17] do, while avoiding the explicit use of state variables and external variables. From a system theoretic point of view, hybrid transition systems are an extension of Sontag machines [39]. Returning to Fig. 1, one might say that the chosen semantics of the original fields are transition systems for computer science, and Sontag machines for system theory. Hybrid transition systems, are our conservative extension of those. On the framework of hybrid transition systems, it turns out to be rather easy to define an operational semantics for actions, as well as for predicates describing flows and re-initializations. Also all kinds of compositions known from process algebra can be defined easily using the method for giving an operational semantics introduced in [40]. As far as we know, HyPA and hybrid χ and the process algebra of [13] are the only process algebras for hybrid systems so far, that use an operational semantics in which complete physical flows are taken into account rather than only the time-behavior of a system.

1.5. Discontinuities

Regarding discontinuous behaviors, the semantics for flow predicates in HyPA, differs a little from the usual interpretation taken in, for example, Henzinger's hybrid automata. The standard approach there (and in most other hybrid formalisms), is to assume only

continuous behavior of all variables, unless they are specifically altered by assignment transitions. For some hybrid descriptions of physical behavior, however, it is convenient that certain variables can also behave discontinuously. Take, for example, the electrical circuit depicted in Fig. 4, in which a switch steers the voltage over a resistor–capacity combination.

For such a system, it is desirable to model the voltage over, and the current through the resistors (v_{R1}, v_{R2}, i_{R1} and i_{R2}) as discontinuous functions of time. A possible hybrid automaton model for this circuit, is depicted in Fig. 5. Note, that there are arbitrary jumps modeled on the transitions, for the discontinuous variables (i.e. not for v_C!). This is necessary, because, without deeper analysis of the differential equations, we do not know what kind of discontinuities may occur. In order to avoid discontinuous behavior that violates the physical properties of the circuit, we may indicate in the hybrid automaton model, that the algebraic equations used to describe the electrical circuit are *invariants*. As an example of an undesired discontinuity, one should note that, when the switch closes, the current through the second resistor (i_{R2}) is determined completely by the source voltage v_e and the voltage over the capacitor v_C. The invariants make sure that no other assignments can be made to i_{R2}.

Now, in the case of higher index differential equations, the approach of using invariants to avoid undesired discontinuities breaks down. As an example, let us consider a system described by the following equations, in which z is a variable that may behave discontinuously:

$$\dot{x} = z,$$
$$\dot{y} = -z,$$
$$x = y.$$

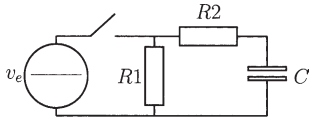

Fig. 4. An electrical circuit with a switch.

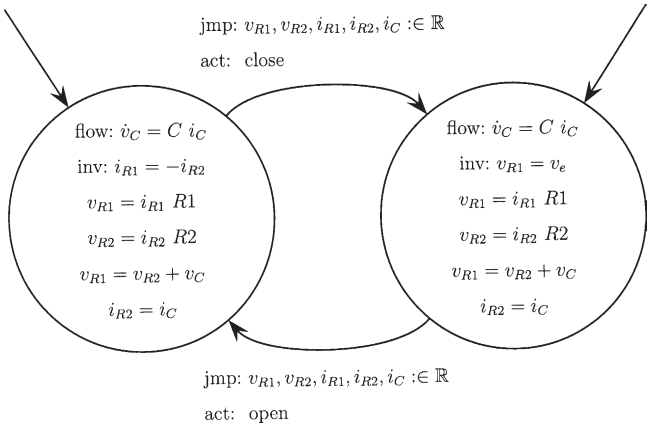

Fig. 5. A hybrid automaton modeling the electrical circuit.

As before, an assignment to z that violates these equations is undesirable. But the approach that is usually taken in hybrid automata theory, to take all algebraic equations to be invariants, does not work here. The choice of z is independent from the choice of x and y. Clearly, the system only can perform continuous behavior, if the value of z is reset immediately to zero. This, however, is insight obtained through analysis of the equations, and should therefore not be used when modeling a system. As far as we know, there is no solution in hybrid automaton theory for this problem. This is why we take a different approach regarding discontinuous behavior in HyPA.

In HyPA, we recognize that differentiated variables can sometimes be discontinuous, and therefore, when modeling a differential equation or other flow predicate, we can indicate explicitly whether a variable is allowed to perform jumps before engaging in a flow. A flow predicate combined with such an indication is called a flow clause. The notation $\left\langle V \mid P_{\mathrm{f}} \right\rangle$, that is formally introduced in the following section, shows a (flow) predicate P_{f}, defining which flows are allowed by the clause, while the set V denotes which variables are not allowed to jump before engaging in a flow. If z is not allowed to jump initially (i.e. $z \in V$), we find deadlock for the higher index differential equations of the previous example when initially $z \neq 0$. If it is allowed to jump ($z \notin V$), only those discontinuities can occur for which a solution exists. Using this way of modeling, the electrical circuit of Fig. 4 could, using HyPA notation, be modeled as the process X in the following equation:

$$X \approx \left(\left\langle v_c \;\middle|\; \begin{array}{l} \dot{v}_C = C\, i_C \\ i_{R1} = -i_{R2} \\ v_{R1} = i_{R1}\, R1 \\ v_{R2} = i_{R2}\, R2 \\ v_{R1} = v_{R2} + v_C \\ i_{R2} = i_C \end{array} \right\rangle \oplus \left\langle v_c \;\middle|\; \begin{array}{l} \dot{v}_C = C\, i_C \\ v_{R1} = v_e \\ v_{R1} = i_{R1}\, R1 \\ v_{R2} = i_{R2}\, R2 \\ v_{R1} = v_{R2} + v_C \\ i_{R2} = i_C \end{array} \right\rangle \right) \rhd X.$$

Notice, that this is not a direct translation of the hybrid automaton. In HyPA, we do not need to give explicit names to the 'open' and 'close' actions, although we could if that were desired from a modeling perspective. Furthermore, it is not necessary to make a distinction between invariants and other flow predicates. In the electrical circuit, the only variable that is not allowed to jump is the voltage over the capacitor. An example in HyPA notation for the higher index system follows shortly.

Assignments in HyPA are modeled, not as a kind of atomic actions (as with hybrid automata), but as re-initializations of processes. These re-initializations can be used as well to model conditional execution of a process. The notation $[V \mid P_{\mathrm{r}}] \gg x$, formally introduced in the following section, denotes that a process x is executed, but with the valuation of the variables changed according to the re-initialization predicate P_{r}. The set V contains, contrary to the notation of flow clauses, those variables that are allowed to change during a re-initialization. For example, an assignment of the value 1 to x, using an action a, under the condition that x is larger than 3 to begin with, is modeled as: $\left[x \mid x^- \geqslant 3 \wedge x^+ = 1\right] \gg a$. Note, that other variables are not allowed to change value while this action is executed. Some peculiar aspects of using re-initialization are discussed in Section 2.2, and sometimes lead to unexpected axioms in Section 3. In the case of our higher index problem, it is possible using axiomatic reasoning, in combination with reasoning on the solutions of differential equations, to obtain the equivalence

$$\left[z \mid z^+ \neq 0\right] \gg \left\langle z \;\middle|\; \begin{array}{l} \dot{x} = z \\ \dot{y} = -z \\ x = y \end{array} \right\rangle \approx \delta,$$

reflecting that an assignment of a value other than 0 to z leads to deadlock, if z is not allowed to jump, and

$$\left[z\middle|\, z^+ \neq 0\right] \gg \begin{pmatrix} \dot{x} = z \\ \dot{y} = -z \\ x = y \end{pmatrix} \approx \begin{pmatrix} \dot{x} = z \\ \dot{y} = -z \\ x = y \end{pmatrix},$$

reflecting that such an assignment is immediately undone if z is allowed to jump. Please note, that this can only be derived if one has a way of calculating with flow-clauses and re-initialization clauses, which is outside the scope of this paper.

1.6. Drawbacks

At first sight, there seem to be two major drawbacks to our method. The first drawback, is that we need a kind of bisimilarity that takes into account the valuation of all variables, in order for it to be a congruence for parallel composition. However, this does not render the whole theory useless, because the same method of requiring compatibility of processes that was used in [17] in order to define parallel composition, can be used in HyPA to guarantee congruence of parallel composition under a weaker notion of equivalence (like the one used in [17]), and furthermore, we give an axiomatization for our notion of equivalence that allows elimination of the parallel composition from closed process terms, so that weaker notions of equivalence can be used for analysis of processes after applying this elimination. The second drawback, is that some of the axioms become rather confusing due to the discontinuities that may be possible in some of the variables of a differential equation. This can be helped, as we show in Section 3, by simply requiring all variables to be continuous, as in hybrid automata. So, in conclusion, the theory is not more difficult or cumbersome, if we model processes under the usual restrictions. In fact, as we indicate in Section 4.1, we expect that HyPA is a conservative extension of hybrid automata, although we do not give a formal proof of this claim. Furthermore, we have new constructs to our disposition that are not available, yet, in other hybrid formalisms, at the cost of having to use more difficult axioms.

Lastly, we have to note that the hybrid process algebra we present is not concerned with any form of abstraction so far, because experience with normal process algebra shows that abstraction is a difficult topic to study algebraically, and we expect it to be convenient, that the basic theory is worked out first [41]. On the other hand, hybrid χ does contain an operator that allows for the hiding of model variables (although there is no axiomatization for it yet), and also the hybrid process algebra of Bergstra and Middelburg [13] has a form of abstraction from model variables. Since the semantics of these languages are comparable to that of HyPA, we expect that it is possible to develop a similar abstraction operator for our language, and hopefully to find a way to reason about it algebraically.

1.7. Structure of this paper

In Section 2.1, the syntax of HyPA is presented, describing how the process algebra ACP [7] is extended with a constant for termination, the so-called disrupt operator, known from LOTOS [8], and variants of the two types of clauses from [2], representing continuous and discontinuous behavior. In Section 2.2, a hybrid transition system semantics is defined in the style of [40], in which continuous behavior is synchronizing, and discrete behavior is interleaving. Section 3 is devoted to an axiomatization of HyPA, for a notion of bisimilarity

[42]. In this section, also the formal relation with ACP is discussed, and a set of basic terms is given into which closed HyPA terms can be rewritten. In Section 4, we give an informal comparison of HyPA with other hybrid formalisms. We conclude by giving our own views on the work presented, and by making suggestions for future research.

2. Hybrid process algebra

2.1. Syntax

In this section, the syntax of HyPA is introduced, which is an extension of the process algebra ACP [7,43], with the disrupt operator from LOTOS [8] and with variants of the flow clauses and re-initialization clauses from the event-flow formalism introduced in [2]. The signature of HyPA consists of the following constant and function symbols:

(1) deadlock δ,
(2) empty process ϵ,
(3) discrete actions $a \in \mathcal{A}$,
(4) flow clauses $c \in C$,
(5) a family of process re-initialization operators $(d \gg \_)_{d \in D}$,
(6) alternative composition $\_ \oplus \_$,
(7) sequential composition $\_ \odot \_$,
(8) disrupt $\_ \blacktriangleright \_$ and left-disrupt $\_ \rhd \_$,
(9) parallel composition $\_ \parallel \_$, left-parallel composition $\_ \between \_$, and forced-synchronization $\_ \mid \_$,
(10) a family of encapsulation operators $(\partial_H(\_))_{H \subseteq \mathcal{A}}$.

The atomic process terms δ (called *deadlock*) and ϵ (called *empty process*) are used to model a deadlocking process and a (successfully) terminating process, respectively. The atomic *discrete actions* are used to model discrete, computational behavior. The set $\mathcal{A}$ of discrete actions is considered a parameter of the theory and can be instantiated at will by the user of our hybrid process algebra.

An atomic flow clause, is a pair $\left\{ V \mid P_{\mathrm{f}} \right\}$ of a set of model variables $V \subseteq \mathcal{V}_{\mathrm{m}}$, signifying which variables are not allowed to jump at the beginning of a flow, and a flow predicate $P_{\mathrm{f}} \in \mathcal{P}_{\mathrm{f}}$ modeling continuous, never terminating, physical behavior. The set of all flow clauses is denoted C. We usually leave out the brackets for V, and even omit it (and the '|' delimiter) if it is empty. Furthermore, the set C is closed under conjunction ($\wedge$) of flow clauses, and using the assumption that there is a flow predicate *false*, which is never satisfied, there is also a flow clause $\left\{ \mathit{false} \right\}$, which is the system theoretic equivalent of deadlock δ. In Section 3, this equivalence is captured in the axiom $\left\{ \mathit{false} \right\} \approx \delta$.

A process re-initialization $d \gg p$ models the behavior of p where the model variables are submitted to a discontinuous change as specified by the re-initialization clause d. A re-initialization clause is a pair $[V \mid P_{\mathrm{r}}]$ of a set of model variables $V \subseteq \mathcal{V}_{\mathrm{m}}$ and a re-initialization predicate P_{r}. The set V models which variables are allowed to change. Note that this is precisely opposite to flow clauses, where V denotes those variables that do not change. The set of all re-initialization clauses is denoted D. The set D is closed under conjunction ($\wedge$), disjunction ($\vee$), and concatenation ($\sim$) of re-initialization clauses. Also, there is a satisfiability operator ($d^?$) on clauses $d \in D$, which does not re-initialize the

values of a model variable, but only executes the re-initialized process, if d can be satisfied in some way. And finally, there is a re-initialization clause (c_{jmp}) derived from a flow clause $c \in C$, which executes the same discontinuities that are allowed initially by the flow clause. These last two operators turn out to be especially useful when calculating with process terms. Using the assumption that there are re-initialization predicates *false* and *true*, we find the process re-initialization $[\![\mathit{false}]\!] \gg p$, executing no behavior since there is no re-initialization satisfying *false*, the process re-initialization $[\mathit{true}] \gg p$, executing exactly the behavior of p, since none of the variables is allowed to change, and the process re-initialization $[\mathcal{V}_{\mathrm{m}} | \mathit{true}] \gg p$, executing p after an arbitrary re-initialization.

The alternative composition $p \oplus q$ models a (non-deterministic) choice between the processes p and q. The sequential composition $p \odot q$ models a sequential execution of processes p and q. The process q is executed after (successful) termination of the process p. We use the notations $\oplus$ and $\odot$ for alternative and sequential composition, rather than the usual $+$ and $\cdot$, to avoid confusion with the notation used frequently in the description of flow and re-initialization predicates for addition and multiplication. We realize that this might distract people in the field of process algebra, yet chose to adapt the process algebraic notation rather than the notation adopted from system theory, simply because the latter has been in use for a longer time already. Overloading the operators is also an option, since it is always clear from the context whether for example addition or choice is intended. When studying HyPA as a new process algebra, as is done in this paper, overloading is probably to be preferred indeed, as it hardly hampers the search for process algebraic properties. However, when studying hybrid models in HyPA, and performing analysis using axioms from both process algebra and system theory in the same proofs, the overloading becomes more of a burden. Furthermore, when presenting these models to other hybrid researchers who are often not familiar with process algebra at all, this effect is even stronger.

The disrupt $p \blacktriangleright q$ models a kind of sequential composition where the process q may take over execution from process p at any moment, without waiting for its termination. This composition is invaluable when modeling two flow clauses executing one after the other, since the behavior of flow clauses is ongoing, and never terminates. The disrupt is originally introduced in the language LOTOS [8], where it is used to model for example exception handling. Also, it is used, for example in [44], for the description of mode switches. The left-disrupt is mainly needed for calculation and axiomatization purposes, rather than for modeling purposes. For example, it occurs often when we attempt to eliminate the parallel composition from a process term through axiomatic reasoning, as described in Section 3. The left-disrupt $p \triangleright q$ first executes a part of the process p and then behaves as a normal disrupt.

The parallel composition $p \parallel q$ models concurrent execution of p and q. The intuition behind this concurrent execution is that discrete actions are executed in an interleaving manner, with the possibility of synchronization (as in ACP, where synchronization is called communication), while flow clauses are forced to synchronize, and can only synchronize if they accept the same solutions. The synchronization of actions takes place using a (partial, commutative, and associative) communication function $\gamma \in \mathcal{A} \times \mathcal{A} \mapsto \mathcal{A}$. For example, if the actions a and a' synchronize, the resulting action is $a'' = a \gamma a'$. Actions cannot synchronize with flow clauses, and in a parallel composition between those, the action executes first. This communication function is considered a parameter of the theory.

As with the left-disrupt, the operators left-parallel composition and forced-communication are mainly introduced for calculation purposes. The left-parallel composition $p \lfloor\!\lfloor q$ models that either p performs a discrete action first, and then behaves as a normal parallel

composition with q, or p cannot perform such an action, and the process deadlocks. The forced-synchronization $p \mid q$ models how the first behavior (either a discrete action or a part of a flow) of p and q is synchronized, after which they behave as in a normal parallel composition. If synchronization is not possible, then the forced-synchronization deadlocks.

Encapsulation $\partial_H(p)$ models that certain discrete actions (from the set $H \subseteq \mathcal{A}$) are blocked during the execution of the process p. This operator is often used in combination with the parallel composition to model that synchronization between discrete actions is enforced.

From the signature of HyPA, terms can be constructed using variables from a given set of process variables $\mathcal{V}_{\mathrm{p}}$ (with $\mathcal{V}_{\mathrm{p}} \cap \mathcal{V}_{\mathrm{m}} = \emptyset$), as usual. In this paper, the set of all such terms is denoted $\mathcal{T}(\mathcal{V}_{\mathrm{p}})$ and these are referred to as terms or open terms. Terms in which no process variables occur are called closed terms. The set of all closed terms is denoted $\mathcal{T}$.

Finally, all the processes should be interpreted in the light of a set E of recursive definitions, called *recursive specification*, of the form $X \approx p$, where X is a process variable and p is a term. We denote the set of all process variables that occur in the left-hand side of a recursive definition from E by $\mathcal{V}_{\mathrm{r}}$ ($\mathcal{V}_{\mathrm{r}} \subseteq \mathcal{V}_{\mathrm{p}}$) and call these variables recursion variables. We only allow recursive definitions $X \approx p$ where the term p only contains recursion variables. Outside the recursive specification, recursion variables are treated as constants of the theory. Recursion is a powerful way to model repetition in a process. We use $X \approx p$ for recursion rather than $X = p$ in order to avoid confusion with equality as used in many syntaxes for describing flow and re-initialization predicates. The set $\mathcal{T}(\mathcal{V}_{\mathrm{r}})$ denotes the set of all terms in which only recursion variables are used. Such elements are referred to as process terms.

The binding order of the operators of HyPA is as follows: $\odot$, $\blacktriangleright$, $\rhd$, $d \gg$, $\|$, $\between$, $|$, $\oplus$, where alternative composition binds weakest, and sequential composition binds strongest. With encapsulation ($\partial_H(\_)$), brackets are always used. As an example, a term $d \gg a \odot b \oplus c \| c'$ should be read as $(d \gg (a \odot b)) \oplus (c \| c')$.

2.2. Formal semantics

In this section, we give a formal semantics to the syntax defined in the previous section, by constructing a kind of labeled transition system, for each process term and each possible valuation of the model variables. In this transition system we consider two different kinds of transitions: one associated with computational behavior (i.e. discrete actions), and the other associated with physical behavior (i.e. flow clauses). This is why we call those transition systems hybrid.

Definition 1 (*Hybrid transition system*). A *hybrid transition system* is a tuple $\langle X, A, \Sigma, \mapsto, \rightsquigarrow, \checkmark \rangle$, consisting of a *state space* X, a *set of action labels* A, a *set of flow labels* Σ, and *transition relations* $\mapsto \subseteq X \times A \times X$ and $\rightsquigarrow \subseteq X \times \Sigma \times X$. Lastly, there is a *termination predicate* $\checkmark \subseteq X$.

For the semantical hybrid transition systems that are associated with HyPA terms, the state space is formed by pairs of process terms and valuations of the model variables, i.e. $X = \mathcal{T}(\mathcal{V}_{\mathrm{r}}) \times \mathit{Val}$. The set of action labels is formed by pairs of actions and valuations, i.e. $A = \mathcal{A} \times \mathit{Val}$, and the set of flow labels is formed by the set of flows, i.e. $\Sigma = \mathcal{F}$.

Recall that the elements $f \in \mathcal{F}$ have a closed-interval domain, possibly a singleton, starting in 0.

We use the notation $\langle x \rangle \overset{a}{\mapsto} \langle x' \rangle$ for a transition $(x, a, x') \in \mapsto$ with $x, x' \in X$ and $a \in A$. Similarly, we use $\langle x \rangle \overset{\sigma}{\rightsquigarrow} \langle x' \rangle$ for a transition $(x, \sigma, x') \in \rightsquigarrow$ with $\sigma \in \Sigma$, and for arbitrary transitions, we use $\langle x \rangle \overset{l}{\rightarrow} \langle x' \rangle$ instead of $(x, l, x') \in \mapsto \cup \rightsquigarrow$ and $l \in A \cup \Sigma$. Finally, termination is denoted $\langle x \rangle \surd$ instead of $x \in \surd$.

Hybrid transition systems [38] can be used to model computational behavior through the use of action transitions, which take no time to execute, and to model physical behavior through the use of flow transitions, which represent the behavior of model variables during the passage of time. Note, that there is no variable in $\mathcal{V}_{\mathrm{m}}$ that is explicitly associated with time. Hence, if one would like to refer to time in a flow clause, one would have to include the model of a clock, using for example a flow clause like $\left[\, t \mid \dot{t} = 1 \right]$.

Before we turn to the actual definition of the semantics of HyPA in terms of hybrid transition systems, a notion of solution for flow clauses and re-initialization clauses is needed for the definition of the semantics of these atoms of the algebra. These notions are obtained by lifting the notion of solution of flow predicates and re-initialization predicates, while taking into account the influence of the variable set V.

A flow clause $[\, V \mid P_{\mathrm{f}}]$ changes the valuation of the model variables according to the possible solutions of its flow predicate P_{f}. In contrast to the flow predicates of [16], an initial jump in the value of a variable x, is allowed in HyPA when $x \notin V$. Furthermore, discontinuous and non-differentiable flows of x may be allowed, if such solutions exists for the type of flow predicate that is used. The concept of solution of a flow clause, is lifted from the notion of solutions of its flow predicate as follows.

Definition 2 (*Solution of a flow clause*). A pair $(\nu, \sigma) \in \mathit{Val} \times \mathcal{F}$, is defined to be a *solution* of a flow clause $c \in C$, denoted $(\nu, \sigma) \models c$, as follows:

- $(\nu, \sigma) \models \left[\, V \mid P_{\mathrm{f}} \right]$ if $\sigma \models_{\mathrm{f}} P_{\mathrm{f}}$, and for all $x \in V$ we find $\nu(x) = \sigma(0)(x)$;
- $(\nu, \sigma) \models c \wedge c'$ if $(\nu, \sigma) \models c$ and $(\nu, \sigma) \models c'$.

Clearly, the flow clause $\left[\mathit{false}\right]$ has no solutions, as the flow predicate *false* has no solutions.

A re-initialization clause $[\, V \mid P_{\mathrm{r}}]$ changes the valuation of the model variables according to the possible solutions of its re-initialization predicate P_{r}. The set V indicates the variables that are allowed to change their value. Whenever $x \notin V$, the variable x is fixed. Note that this is precisely opposite to the use of V in flow clauses. We define the solutions of a re-initialization clause in terms of the solutions of a re-initialization predicate as follows.

Definition 3 (*Solution of a re-initialization clause*). A re-initialization $(\nu, \nu') \in \mathcal{R}$ is defined to be a *solution* of a re-initialization clause $d \in D$, denoted $(\nu, \nu') \models d$, as follows:

- $(\nu, \nu') \models [\, V \mid P_{\mathrm{r}}]$ if $(\nu, \nu') \models_{\mathrm{r}} P_{\mathrm{r}}$ and for all $x \notin V$ we find $\nu(x) = \nu'(x)$;
- $(\nu, \nu') \models d' \vee d''$ if $(\nu, \nu') \models d'$ or $(\nu, \nu') \models d''$;
- $(\nu, \nu') \models d' \wedge d''$ if $(\nu, \nu') \models d'$ and $(\nu, \nu') \models d''$;
- $(\nu, \nu') \models d' \sim d''$ if there exists $\upsilon \in \mathit{Val}$ with $(\nu, \upsilon) \models d'$ and $(\upsilon, \nu') \models d''$;
- $(\nu, \nu') \models d'^{?}$ if $\nu = \nu'$, and there exists $\upsilon \in \mathit{Val}$ with $(\nu, \upsilon) \models d'$;
- $(\nu, \nu') \models c_{\mathit{jmp}}$ if there exists $\sigma \in \Sigma$ such that $(\nu, \sigma) \models c$ and $\sigma(0) = \nu'$.

If we have two re-initialization clauses $d, d' \in D$, the clause $d \sim d'$ accepts exactly those solutions that are a concatenation of the re-initializations of d and d'. The clause $d^?$ does not change the value of any of the variables, and only has a solution for those valuations for which d has a solution. The clause c_{jmp} imitates the re-initializations performed initially by a flow clause c. Obviously, the re-initialization clause $[\mathit{false}]$ has no solutions, while $[\mathcal{V}_\mathrm{m}|\, \mathit{true}]$ has every possible re-initialization as a solution. Note, that $[\mathit{true}]$ exactly allows all re-initializations that do not change any of the variable valuations.

The semantics of the HyPA constants and function symbols is given in Tables 1–5, using deduction rules in the style of [40]. In these tables p, p', q, q' denote process terms, a, a', a'' denote actions, c denotes a flow clause, d denotes a re-initialization clause, H denotes a set of actions, X denotes a recursion variable, ν, ν', ν'' denote valuations, σ denotes a flow, t denotes a point in time, and l denotes an arbitrary transition label.

In Table 1, the semantics of the atomic processes, the flow clauses, and the process re-initializations is given. Rule (1) captures our intuition that ϵ is a process that only terminates. Analogously, the fact that there is no rule for δ, expresses that this is indeed a deadlocking process. Rule (2) expresses that discrete actions display their own name, and the valuation of the model variables on the transition label, but do not change this valuation. Changes in the valuation can only be caused by flow clauses and re-initialization clauses, as defined by rules (3)–(5).

The semantics of the other operators is defined in Tables 2–5. Rules (6)–(10), for alternative and sequential composition, are very similar to that of ACP. However, it is worth noting that we have chosen to model flow transitions as having the same non-deterministic interpretation as action transitions. This in contrast to many timed process algebras [45], where the passage of time (by itself) does not trigger a branching in the transition system. The reason for this way of modeling, is our intuition that continuous behavior (i.e. the

Table 1
Operational semantics of HyPA

$$\frac{}{\langle\, \epsilon, \nu \,\rangle \surd}(1) \qquad \frac{}{\langle\, a, \nu \,\rangle \overset{a,\nu}{\mapsto} \langle\, \epsilon, \nu \,\rangle}(2) \qquad \frac{(\nu, \sigma) \models c,\ \mathit{dom}(\sigma) = [0, t]}{\langle\, c, \nu \,\rangle \overset{\sigma}{\rightsquigarrow} \langle\, c, \sigma(t) \,\rangle}(3)$$

$$\frac{(\nu, \nu') \models d,\ \langle\, p, \nu' \,\rangle \surd}{\langle\, d \gg p, \nu \,\rangle \surd}(4) \qquad \frac{(\nu, \nu') \models d,\ \langle\, p, \nu' \,\rangle \overset{l}{\to} \langle\, p', \nu'' \,\rangle}{\langle\, d \gg p, \nu \,\rangle \overset{l}{\to} \langle\, p', \nu'' \,\rangle}(5)$$

Table 2
Operational semantics of HyPA, alternative and sequential composition

$$\frac{\langle\, p, \nu \,\rangle \surd}{\begin{array}{l}\langle\, p \oplus q, \nu \,\rangle \surd \\ \langle\, q \oplus p, \nu \,\rangle \surd\end{array}}(6) \qquad \frac{\langle\, p, \nu \,\rangle \overset{l}{\to} \langle\, p', \nu' \,\rangle}{\begin{array}{l}\langle\, p \oplus q, \nu \,\rangle \overset{l}{\to} \langle\, p', \nu' \,\rangle \\ \langle\, q \oplus p, \nu \,\rangle \overset{l}{\to} \langle\, p', \nu' \,\rangle\end{array}}(7) \qquad \frac{\langle\, p, \nu \,\rangle \surd,\ \langle\, q, \nu \,\rangle \surd}{\langle\, p \odot q, \nu \,\rangle \surd}(8)$$

$$\frac{\langle\, p, \nu \,\rangle \overset{l}{\to} \langle\, p', \nu' \,\rangle}{\langle\, p \odot q, \nu \,\rangle \overset{l}{\to} \langle\, p' \odot q, \nu' \,\rangle}(9) \qquad \frac{\langle\, p, \nu \,\rangle \surd,\ \langle\, q, \nu \,\rangle \overset{l}{\to} \langle\, q', \nu' \,\rangle}{\langle\, p \odot q, \nu \,\rangle \overset{l}{\to} \langle\, q', \nu' \,\rangle}(10)$$

Table 3
Operational semantics of HyPA, disrupt

$$\frac{\langle p, \nu \rangle \checkmark}{\begin{array}{l}\langle p \blacktriangleright q, \nu \rangle \checkmark \\ \langle p \rhd q, \nu \rangle \checkmark\end{array}}(11) \qquad \frac{\langle p, \nu \rangle \xrightarrow{l} \langle p', \nu' \rangle}{\begin{array}{l}\langle p \blacktriangleright q, \nu \rangle \xrightarrow{l} \langle p' \blacktriangleright q, \nu' \rangle \\ \langle p \rhd q, \nu \rangle \xrightarrow{l} \langle p' \blacktriangleright q, \nu' \rangle\end{array}}(12)$$

$$\frac{\langle q, \nu \rangle \checkmark}{\langle p \blacktriangleright q, \nu \rangle \checkmark}(13) \qquad \frac{\langle q, \nu \rangle \xrightarrow{l} \langle q', \nu' \rangle}{\langle p \blacktriangleright q, \nu \rangle \xrightarrow{l} \langle q', \nu' \rangle}(14)$$

Table 4
Operational semantics of HyPA, parallel composition

$$\frac{\langle p, \nu \rangle \checkmark, \langle q, \nu \rangle \checkmark}{\begin{array}{l}\langle p \parallel q, \nu \rangle \checkmark \\ \langle p \mid q, \nu \rangle \checkmark\end{array}}(15) \qquad \frac{\langle p, \nu \rangle \overset{\sigma}{\rightsquigarrow} \langle p', \nu' \rangle,\ \langle q, \nu \rangle \overset{\sigma}{\rightsquigarrow} \langle q', \nu' \rangle}{\begin{array}{l}\langle p \parallel q, \nu \rangle \overset{\sigma}{\rightsquigarrow} \langle p' \parallel q', \nu' \rangle \\ \langle p \mid q, \nu \rangle \overset{\sigma}{\rightsquigarrow} \langle p' \parallel q', \nu' \rangle\end{array}}(16)$$

$$\frac{\langle p, \nu \rangle \overset{\sigma}{\rightsquigarrow} \langle p', \nu' \rangle,\ \langle q, \nu \rangle \checkmark}{\begin{array}{l}\langle p \parallel q, \nu \rangle \overset{\sigma}{\rightsquigarrow} \langle p', \nu' \rangle \\ \langle q \parallel p, \nu \rangle \overset{\sigma}{\rightsquigarrow} \langle p', \nu' \rangle \\ \langle p \mid q, \nu \rangle \overset{\sigma}{\rightsquigarrow} \langle p', \nu' \rangle \\ \langle q \mid p, \nu \rangle \overset{\sigma}{\rightsquigarrow} \langle p', \nu' \rangle\end{array}}(17) \qquad \frac{\langle p, \nu \rangle \overset{a,\nu'}{\mapsto} \langle p', \nu'' \rangle}{\begin{array}{l}\langle p \parallel q, \nu \rangle \overset{a,\nu'}{\mapsto} \langle p' \parallel q, \nu'' \rangle \\ \langle q \parallel p, \nu \rangle \overset{a,\nu'}{\mapsto} \langle q \parallel p', \nu'' \rangle \\ \langle p \lfloor\!\lfloor q, \nu \rangle \overset{a,\nu'}{\mapsto} \langle p' \parallel q, \nu'' \rangle\end{array}}(18)$$

$$\frac{\langle p, \nu \rangle \overset{a,\nu'}{\mapsto} \langle p', \nu'' \rangle,\ \langle q, \nu \rangle \overset{a',\nu'}{\mapsto} \langle q', \nu'' \rangle,\ a'' = a \,\gamma\, a'}{\begin{array}{l}\langle p \parallel q, \nu \rangle \overset{a'',\nu'}{\mapsto} \langle p' \parallel q', \nu'' \rangle \\ \langle p \mid q, \nu \rangle \overset{a'',\nu'}{\mapsto} \langle p' \parallel q', \nu'' \rangle\end{array}}(19)$$

Table 5
Operational semantics of HyPA, encapsulation and recursion

$$\frac{\langle p, \nu \rangle \overset{a,\nu'}{\mapsto} \langle p', \nu'' \rangle,\ a \notin H}{\langle \partial_H(p), \nu \rangle \overset{a,\nu'}{\mapsto} \langle \partial_H(p'), \nu'' \rangle}(20)$$

$$\frac{\langle p, \nu \rangle \overset{\sigma}{\rightsquigarrow} \langle p', \nu' \rangle}{\langle \partial_H(p), \nu \rangle \overset{\sigma}{\rightsquigarrow} \langle \partial_H(p'), \nu' \rangle}(21) \qquad \frac{\langle p, \nu \rangle \checkmark}{\langle \partial_H(p), \nu \rangle \checkmark}(22)$$

$$\frac{\langle p, \nu \rangle \checkmark}{\langle X, \nu \rangle \checkmark}(23)\ X \approx p \in E \qquad \frac{\langle p, \nu \rangle \xrightarrow{l} \langle p', \nu' \rangle}{\langle X, \nu \rangle \xrightarrow{l} \langle p', \nu' \rangle}(24)\ X \approx p \in E$$

passing of time) influences the valuation of the model variables, and can therefore introduce choices in the system behavior, just like discrete actions do. If, in the future, we develop operators to abstract from the variables that trigger those choices, we do not want the choices themselves to disappear, through some time-determinism mechanism. The argument for introducing time-determinism, that time is an external phenomenon that does not influence the state of a system, does in our opinion not hold for hybrid systems. Also, the hybrid automata of Henzinger [16], and most other hybrid automata approaches that we know of, are time-non-deterministic, supposedly for the same reasons.

Interestingly, in [13] a time-deterministic approach to hybrid systems is chosen (clearly, they disagree with the above arguments), while in hybrid χ [11] operators are introduced for both. Models in the language hybrid χ, therefore, might show the difference between the approaches. As far as we can tell, the time-deterministic operator is used most often when, for example, a controller makes a choice after some delay, indeed without specifying the dynamics during this delay. This is modeled as a time-deterministic choice between delaying actions. When modeling physical modes of a system, the non-deterministic choice operator is used. The physical behavior of a system can only be in one mode, even if a particular evolution is permitted in both modes. In other words, time-determinism plays a role on a higher level of abstraction than that which we aim for in HyPA.

Rules (11)–(14) define the semantics of the disrupt operator and the left-disrupt operator. If we compare these rules to the rules for sequential composition, we see that the main difference, is the way in which termination is handled. Firstly, in a composition $p \blacktriangleright q$, the process q may start execution without p terminating. Secondly, if the process p terminates, the process $p \blacktriangleright q$ may also terminate regardless of the behavior of q.

Rules (15)–(19) define the semantics of the parallel composition, and in these rules the difference between action transitions and flow transitions is most prominent. For actions, the interpretation of the parallel composition is the same as in ACP [7,43]. Discrete actions that are placed in parallel are interleaved, but can also synchronize using a (partial, commutative, and associative) communication function $\gamma \in \mathcal{A} \times \mathcal{A} \mapsto \mathcal{A}$. If a discrete action a communicates with an action a' (this is the case if $a\gamma a'$ is defined), the result is an action $a'' = a\gamma a'$. If flow clauses are placed in parallel, they always synchronize their behavior such that, intuitively, the flows that are possible in a parallel composition are a solution of both clauses.

Encapsulation, as defined by rules (20)–(22), only influences action transitions. This is not surprising, since, as mentioned before, the $\partial_H(\_)$ operator is originally intended to model enforced synchronization in a parallel composition. Parallel composition, in general, may lead to interleaving actions and synchronized actions. The encapsulation operator is then used to block the interleaving actions. Flow transitions are already synchronized in the parallel composition, so there is no need for encapsulation of those.

Rules (23) and (24) model recursion in the same way as it was done in [7,43]. For a recursive definition $X \approx p$, a transition for the variable X is possible, if it can be deduced from the semantical rules for the process term p.

2.3. Bisimilarity

In this section, we discuss the equivalence notion of bisimilarity [42], which is first defined on hybrid transition systems, and then lifted to process terms.

Definition 4 (*Bisimilarity on hybrid transition systems*). Given, a hybrid transition system $\langle X, A, \Sigma, \mapsto, \rightsquigarrow, \checkmark \rangle$, a relation $R \subseteq X \times X$ is a *bisimulation* relation if

- for all $x, y \in X$ such that xRy, we find $\langle x \rangle \checkmark$ implies $\langle y \rangle \checkmark$;
- for all $x, y \in X$ such that xRy, we find $\langle y \rangle \checkmark$ implies $\langle x \rangle \checkmark$;
- for all $x, x', y \in X$ such that xRy and $l \in A \cup \Sigma$, we find $\langle x \rangle \xrightarrow{l} \langle x' \rangle$ implies there exists y' such that $\langle y \rangle \xrightarrow{l} \langle y' \rangle$ and $x'Ry'$;
- for all $x, y, y' \in X$ such that xRy and $l \in A \cup \Sigma$, we find $\langle y \rangle \xrightarrow{l} \langle y' \rangle$ implies there exists x' such that $\langle x \rangle \xrightarrow{l} \langle x' \rangle$ and $x'Ry'$.

Two states $x, y \in X$ are *bisimilar*, notation $x \leftrightarrows y$, if there exists a bisimulation relation that relates x and y.

In lifting this notion of equivalence on hybrid transition systems to process terms (and hence abstracting from valuations) we have to be careful. It is assumed that the model variables that are shared by the process terms to be related represent the same entity. Therefore, both process terms are only compared with respect to the same (arbitrary) initial valuation of the model variables.

In order for the equivalence to be robust with respect to interference caused by processes executed in parallel, for all states that are reached by performing transitions, it is required that the contained process terms are related for all valuations that can be obtained through interference. This is what we call robustness of a relation. An interference can be modeled as a function $\iota : Val \to Val$. Observe that we apply the same interference function to both variable valuations.

Definition 5 (*Robust*). A relation $R \subseteq (\mathcal{T}(\mathcal{V}_r) \times Val) \times (\mathcal{T}(\mathcal{V}_r) \times Val)$ is *robust* if for all $\langle p, \nu \rangle, \langle p', \nu' \rangle \in X$ such that $\langle p, \nu \rangle R \langle p', \nu' \rangle$, and for all interferences $\iota \in Val \to Val$, we find $\langle p, \iota(\nu) \rangle R \langle p', \iota(\nu') \rangle$.

Definition 6 (*Robust bisimilarity*). Two process terms $p, q \in \mathcal{T}(\mathcal{V}_r)$ are robustly bisimilar, denoted $p \leftrightarrows q$, if there exists a robust bisimulation relation $R \subseteq (\mathcal{T}(\mathcal{V}_r) \times Val) \times (\mathcal{T}(\mathcal{V}_r) \times Val)$ such that $\langle p, \nu \rangle R \langle q, \nu \rangle$ for all valuations $\nu \in Val$.

If two process terms are robustly bisimilar, then they describe equivalent transition systems, hence they describe the same process. In Appendix B, we show that the notion of robust bisimilarity given here coincides with the notion of bisimilarity (also called stateless bisimilarity) used in [46]. We adapted the definition, because it separates the idea of interference from the notion of bisimilarity. In the following section, we discuss an axiomatization of robust bisimilarity for HyPA. First, however, we give two examples of modeling in HyPA, in order to strengthen the intuition on its syntax and semantics.

2.4. Example: steam boiler

This section is intended to illustrate the use of HyPA for modeling hybrid systems. The process below, is a model of the celebrated benchmark problem of the steam boiler [47]. For reasons of brevity, the problem is simplified considerably. It is not our intention to give a comparison with other models of the steam boiler here. We only want to give a feeling

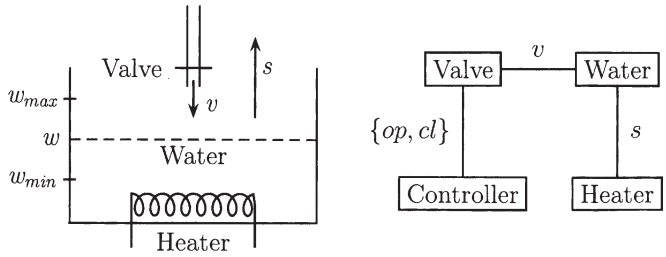

Fig. 6. The steam boiler.

for the syntax and semantics of the language. The text below, explains shortly what the given model consists of.

The boiler process, as depicted in Fig. 6 consists of a water level w, an in-flow of water v and a steam production s. This steam production is determined by the Heater process, which limits it between the constants s_{min} and s_{max}. The in-flow is determined by a Valve process, which can be opened or closed using the actions *ro* and *rc* respectively. If the valve is open, the in-flow to the boiler is v_{in}. If it is closed, the in-flow is 0. Furthermore, there is a Controller, that every T seconds interferes with the valve, by telling it to open or close using the actions *so* and *sc*. The goal of this controller, is to keep the water level between the constants w_{min} and w_{max}. To do this safely, it takes a margin of w_{safe} into account. The total system is the parallel composition of the Water process, the Heater, the two modes of the Valve, and the Controller, over which communication is enforced through the definitions $op = ro\ \gamma\ so$, $cl = rc\ \gamma\ sc$, and $H = \{so, sc, ro, rc\}$:

$$
\begin{aligned}
\text{Water} &\approx \left(w \mid \dot{w} = v - s \right),\\
\text{Heater} &\approx \left(s_{min} \leqslant s \leqslant s_{max} \right),\\
\text{ValveOpen} &\approx \left(v = v_{in} \right) \blacktriangleright rc \odot \text{ValveClose},\\
\text{ValveClose} &\approx \left(v = 0 \right) \blacktriangleright ro \odot \text{ValveOpen},\\
\text{Controller} &\approx \left[t \mid t^{+} = 0 \right] \gg \left(t \;\middle|\; \begin{matrix} \dot{t} = 1 \\ t \leqslant T \end{matrix} \right) \blacktriangleright \left[t^{-} = T \right] \gg\\
&\qquad \left(\begin{matrix} \left[w^{-} \geqslant w_{max} - w_{safe} \right] \gg sc \odot \text{Controller} \oplus \\ \left[w_{min} + w_{safe} \leqslant w^{-} \leqslant w_{max} - w_{safe} \right] \gg \text{Controller} \oplus \\ \left[w^{-} \leqslant w_{min} + w_{safe} \right] \gg so \odot \text{Controller} \end{matrix} \right),\\
\text{Boiler} &\approx \partial_H \left(\text{Water} \parallel \text{Heater} \parallel (\text{ValveOpen} \oplus \text{ValveClosed}) \parallel \text{Controller} \right).
\end{aligned}
$$

In the following section, we discuss an axiomatization of HyPA that allows us to rewrite the Boiler process into a form in which all parallel compositions are eliminated.

2.5. Example: impact control

In this section, we give another modeling example. This time, of a system in which the discontinuities are also governed by physical laws. Fig. 7 shows a part of a component mounter, that places a component, modeled as a simple mass M_c driven by a force f_c, onto a printed circuit board (PCB), modeled as a mass M_p, in connection with a spring-damper system K_p, R_p to reflect the flexibility of the board. From a hybrid point of view, mainly

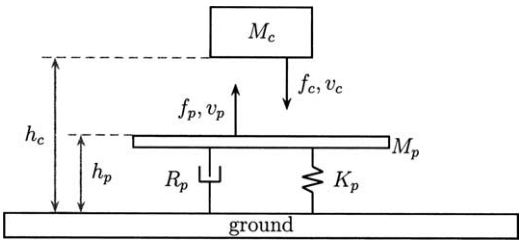

Fig. 7. Schematic model of the impact process.

the moment of impact is interesting, because it can be modeled using laws of conservation of momentum.

Regarding the movement of the masses, two distinct phases can be recognized. In the one phase, where the component and the PCB do not touch, the movement of the component is determined by the force f_c, while the movement of the PCB is determined by the internal force f_p of the spring-damper system. In the other phase, where the component touches the PCB, the total of the forces is divided over the total of the masses. Note, that in one mode the velocities are not allowed to jump, while in the other mode they are. There seems to be a relation between this phenomenon and the 'loss of state' described in [1], due to dependencies between masses or other energy storing elements. Also note, that we use a left-disrupt rather than a disrupt operator, in order to make the definition 'guarded'. This is explained in Section 3. We find the following model for the movement of the masses:

$$\mathrm{M} \approx \left(\left[\begin{array}{c|l} h_c, v_c \\ h_p, v_p & \begin{array}{l} h_c \geqslant h_p \\ f_c = M_c \dot{v}_c \\ f_p = M_p \dot{v}_p \\ \dot{h}_c = v_c \\ \dot{h}_p = v_p \end{array} \end{array} \right] \oplus \left[\begin{array}{c|l} h_c \\ h_p & \begin{array}{l} f_c M_p \geqslant f_p M_c \\ h_c = h_p \\ f_c + f_p = M_c \dot{v}_c + M_p \dot{v}_p \\ \dot{h}_c = v_c \\ \dot{h}_p = v_p \end{array} \end{array} \right] \right) \rhd \mathrm{M}.$$

Of course, the force on the PCB is determined by the spring-damper system. This is modeled by the following process:

$$\mathrm{SD} \approx \left[\; h_p \; \middle| \; f_p = -K_p h_p - R_p v_p \; \right].$$

The model of movement, however, needs to be restricted, since during the collision very wild jumps in the velocities are possible. We need to pose laws for the conservation of momentum, and the conservation (or rather the decrease) of energy in the system. The following process models that the change in total momentum p depends on the total force on the masses, while this total momentum is not allowed to jump. The total momentum is the derivative of the total energy e in the system, and the total energy may only decrease during jumps:

$$\mathrm{E} \approx \left[e \middle| e^+ \leqslant e^- \right] \gg \left[\begin{array}{c|l} p & \begin{array}{l} \dot{p} = f_c + f_p \\ \dot{e} = p \\ p = M_c v_c + M_p v_p \end{array} \end{array} \right] \rhd \mathrm{E}.$$

The total model of the component mounter then becomes M ‖ SD ‖ E. One of the problems we are faced with at the moment, is to find a suitable controller for this process, steering the force f_c in such a way that the total play of forces and the change in velocity at impact, never become such that the component cracks. In other words, to find a process C such that M ‖ SD ‖ E ‖ C has some desired safety properties. This is a topic for future research.

3. Algebraic reasoning in HyPA

The strength of the field of process algebra, lies in its ability to use equational reasoning for the analysis of transition systems, or, more precisely, for the analysis of equivalence classes of transition systems, called processes. In this section, we show that this equational reasoning is also possible in HyPA. In the previous section, a notion of bisimilarity was defined on process terms, reflecting equivalence of the underlying hybrid transition systems. We study properties of this equivalence, and capture those properties in a set of *derivation rules* and a set of *axioms* on the algebra of process terms. Together with a *principle for guarded recursion*, this forms a proof system in which every derived equality on process terms represents equality of the underlying hybrid transition systems. In other words, process terms that are derivably equal, describe transition systems in the same equivalence class, and hence describe the same process.

This section is split up in four parts. In the first part, we give a formal axiomatization of robust bisimilarity, and we treat the intuition behind the axioms, and the insights they provide us with. In the second part, we prove soundness of this axiomatization. In the third part, we discuss a specification principle that is used for reasoning about recursion, and in the fourth part, we show a few useful properties of our axiomatization, like a conservativity theorem with respect to the process algebra ACP and a rewrite system for rewriting closed terms into a normal form.

3.1. Axiomatization

In this subsection, we give the axiomatization of robust bisimilarity in HyPA. In Table 6, we give a set of derivation rules, and throughout this subsection we give a set of axioms that, to a large extend, capture the notion of robust bisimilarity. We write HyPA $\vdash_E p \approx q$, if we can derive equivalence of p and q using those axioms and recursive definitions from a set E.

Definition 7 (*Derivation*). Let E be a set of recursive definitions over a set of recursion variables $\mathcal{V}_\mathrm{r}$. We write HyPA $\vdash_E p \approx q$ to indicate that equivalence of (open) terms p and q can be *derived* from our axiom system and the recursive definitions from E. We define that equivalence can be derived according to the rules given in Table 6. In this table, p, p_i, q, q_i, r denote terms, d, d' denote re-initialization clauses, and c, c', c'' denote flow clauses.

In cases where there can be no confusion as to the set of recursive definitions that is intended, we write $\vdash$ instead of $\vdash_E$.

In the remainder of this subsection, the axioms of HyPA, and the insight they provide regarding the operators of the language, are presented. Also, the intuitions behind derivation rules (9) and (10), are discussed. In each of the axioms, x, y, z denote arbitrary terms. The letters a, a' denote actions, while c, c' denote flow clauses and d, d' denote re-initialization clauses. Unlike what is usual for ACP, one may not choose δ when a is written in an axiom.

The first five axioms, known as the axioms of *basic process algebra* [7], model properties of choice and sequential composition. Alternative composition is idempotent, because a choice between equals is not really considered a choice. Furthermore, it is associative and commutative. Sequential composition is only associative. Sequential composition

Table 6
Derivation rules of HyPA

$$\frac{}{\text{HyPA} \vdash_E p \approx p}(1) \quad \frac{\text{HyPA} \vdash_E p \approx q}{\text{HyPA} \vdash_E q \approx p}(2)$$

$$\frac{\text{HyPA} \vdash_E p \approx q,\ \text{HyPA} \vdash_E q \approx r}{\text{HyPA} \vdash_E p \approx r}(3)$$

$$\frac{\text{HyPA} \vdash_E p \approx q,\ S : \mathcal{V}_{\mathrm{p}} \to \mathcal{T}(\mathcal{V}_{\mathrm{p}}),\ \mathrm{dom}(S) \cap \mathcal{V}_{\mathrm{r}} = \emptyset}{\text{HyPA} \vdash_E S(p) \approx S(q)}(4)$$

$$\frac{\mathcal{O} \text{ an n-ary HyPA operator},\ \forall_{1 \leqslant i \leqslant n}\ \text{HyPA} \vdash_E p_i \approx q_i}{\text{HyPA} \vdash_E \mathcal{O}(p_1, \ldots, p_n) \approx \mathcal{O}(q_1, \ldots, q_n)}(5)$$

$$\frac{\forall_{\nu,\nu'}\ (\nu, \nu') \models d \text{ iff } (\nu, \nu') \models d'}{\text{HyPA} \vdash_E d \gg x \approx d' \gg x}(6) \quad \frac{\forall_{\nu,\sigma}\ (\nu, \sigma) \models c \text{ iff } (\nu, \sigma) \models c'}{\text{HyPA} \vdash_E c \approx c'}(7)$$

$$\frac{p \approx q \text{ is an axiom or a recursive definition}}{\text{HyPA} \vdash_E p \approx q}(8)$$

$$\frac{\begin{array}{c}\forall_{\nu,\sigma}\ (\nu, \sigma) \models c' \text{ implies } (\nu, \sigma) \models c \\ \forall_{\nu,\nu',\sigma}\ (\nu, \nu') \models d \text{ and } (\nu', \sigma) \models c \text{ imply } (\nu', \sigma) \models c'\end{array}}{\text{HyPA} \vdash_E d \gg c \approx d \gg c' \rhd c}(9)$$

$$\frac{\forall_{\nu,\sigma}\ (\nu, \sigma) \models c \text{ iff } (\nu, \sigma) \models c' \text{ or } (\nu, \sigma) \models c''}{\text{HyPA} \vdash_E c \approx (c' \oplus c'') \rhd c}(10)$$

right-distributes over alternative composition, but does not left-distribute since that would lead to a change in the moment of choice.

$$\begin{array}{|l|}\hline x \oplus y \approx y \oplus x \\ (x \oplus y) \oplus z \approx x \oplus (y \oplus z) \\ x \oplus x \approx x \\ (x \oplus y) \odot z \approx x \odot z \oplus y \odot z \\ (x \odot y) \odot z \approx x \odot (y \odot z) \\ \hline\end{array}$$

Alternative composition and sequential composition have deadlock and the empty process, respectively, as a unit element, while deadlock is a left-zero element for sequential composition.

$$\begin{array}{|llll|}\hline x \oplus \delta \approx x & \epsilon \odot x & \approx x \\ \delta \odot x \approx \delta & x \odot \epsilon & \approx x \\ \hline\end{array}$$

In fact, any flow-clause is a left-zero element of sequential composition, since flow-clauses do not terminate. This is a generalization of the previous axiom, recalling a previous remark that $\left\lfloor false \right\rfloor$ is the system theoretic equivalent of deadlock. For many of the operators, the role of deadlock can be derived from the axioms on flow clauses. Often, however, we give the axiom for deadlock separately, for sake of clarity of the presentation.

$$\begin{array}{|lll|}\hline c \odot x \approx c & \left\lfloor false \right\rfloor & \approx \delta \\ \hline\end{array}$$

The disrupt operator, can only be axiomatized using the left-disrupt (see [44]).

$$x \blacktriangleright y \approx x \rhd y \oplus y$$

For the left-disrupt, we find the following axioms, that reflect a kind of associativity, and right-distribution over the alternative composition. Deadlock is a left-zero, but a right-unit element of the left-disrupt. Also, there are two axioms formalizing the relation between sequential composition and left-disrupt. The last of these axioms reflects that, if the left argument of the left-disrupt does not terminate, then sequential composition distributes over left-disrupt. Derivation rules (9) and (10), which also deal with the left-disrupt, are discussed at the end of this section.

$$\begin{array}{cc}
(x \rhd y) \rhd z \approx x \rhd (y \blacktriangleright z) & (x \oplus y) \rhd z \approx x \rhd z \oplus y \rhd z \\
\delta \rhd x \approx \delta & x \rhd \delta \approx x \\
a \odot x \rhd y \approx a \odot (x \blacktriangleright y) & \epsilon \rhd x \approx \epsilon \\
(x \odot \delta \rhd y) \odot z \approx x \odot \delta \rhd y \odot z &
\end{array}$$

The axiomatization of parallel composition relies on the axiomatization of the left-parallel composition and the forced-synchronization operator.

$$x \parallel y \approx x \lfloor\!\lfloor y \oplus y \lfloor\!\lfloor x \oplus x \mid y$$

Regarding the left-parallel composition and the forced-synchronization, we find the following axioms that describe associativity and commutativity properties. The axioms also describe how all independent behavior of parallel composition is executed by the left-parallel composition, while synchronization amongst actions, amongst flow-clauses and between termination and flow-clauses is executed by the forced-synchronization operator. Note, that this corresponds to the choice made in [48], and is subtly different from the way parallel composition is treated in [7]. For the forced-synchronization operator, we find termination if both the left and the right process terminate. Termination cannot synchronize with actions, and therefore leads to deadlock. Actions a and a' may synchronize by producing an action $a\gamma a'$ if this action is defined, and otherwise the forced-synchronization results in deadlock. Termination may occur before flow behavior executes, actions and flows cannot synchronize, and flows always must synchronize.

$$\begin{array}{cc}
\epsilon \lfloor\!\lfloor x \approx \delta & x \parallel \epsilon \approx x \\
a \odot x \lfloor\!\lfloor y \approx a \odot (x \parallel y) & (x \oplus y) \lfloor\!\lfloor z \approx x \lfloor\!\lfloor z \oplus y \lfloor\!\lfloor z \\
c \rhd x \lfloor\!\lfloor y \approx \delta & (x \lfloor\!\lfloor y) \lfloor\!\lfloor z \approx x \lfloor\!\lfloor (y \parallel z) \\
\delta \lfloor\!\lfloor x \approx \delta & (x \mid y) \lfloor\!\lfloor z \approx x \mid (y \lfloor\!\lfloor z) \\
\epsilon \mid \epsilon \approx \epsilon & x \mid y \approx y \mid x \\
\delta \mid x \approx \delta & (x \oplus y) \mid z \approx x \mid z \oplus y \mid z \\
\epsilon \mid c \rhd x \approx c \rhd x & (x \mid y) \mid z \approx x \mid (y \mid z) \\
\epsilon \mid a \odot x \approx \delta &
\end{array}$$

$$\begin{array}{l}
a \odot x \mid b \odot y \approx (a\gamma b) \odot (x \parallel y) \text{ if } (a\gamma b) \text{ defined} \\
a \odot x \mid b \odot y \approx (a\gamma b) \odot (x \parallel y) \text{ if } (a\gamma b) \text{ undefined} \\
a \odot x \mid c \rhd y \approx \delta \\
c \rhd x \mid c' \rhd y \approx (c \wedge c') \rhd \begin{pmatrix} x \lfloor\!\lfloor c' \blacktriangleright y \oplus \\ y \lfloor\!\lfloor c \blacktriangleright x \oplus \\ x \mid c' \blacktriangleright y \oplus \\ y \mid c \blacktriangleright x \end{pmatrix}
\end{array}$$

Notice, that the axioms on left-disrupt, left-parallel composition and forced-synchronization may be used to prove additional equalities, such as $(x \blacktriangleright y) \blacktriangleright z \approx x \blacktriangleright (y \blacktriangleright z)$, $x \parallel y \approx y \parallel x$, and $(x \parallel y) \parallel z \approx x \parallel (y \parallel z)$.

As usual, encapsulation of actions distributes over all operators, except over parallel composition, left-parallel composition and forced-synchronization.

$\partial_H(c) \approx c$	$\partial_H(x \oplus y) \approx \partial_H(x) \oplus \partial_H(y)$
$\partial_H(\epsilon) \approx \epsilon$	$\partial_H(x \odot y) \approx \partial_H(x) \odot \partial_H(y)$
$\partial_H(\delta) \approx \delta$	$\partial_H(x \rhd y) \approx \partial_H(x) \rhd \partial_H(y)$
$\partial_H(a) \approx a$ if $a \notin H$	$\partial_H(a) \approx \delta$ if $a \in H$

Finally, we should pay attention to the re-initialization operator. There are re-initialization clauses $[true]$, serving as a unit element, and $\lfloor false \rfloor$, serving as an equivalent for deadlock. Deadlock itself a zero-element. Furthermore, subsequent re-initializations can be concatenated using the $\sim$ operation, and a flow-clause c has an implicit re-initialization c_{jmp}, modeling spontaneous re-initializations following from initial value problems as described in, for example, [1]. The re-initialization operator distributes over most other operators from HyPA, except over the parallel composition and the forced-synchronization. With respect to termination, re-initialization has peculiar behavior. Because a re-initialization d is executed at the beginning of the first transition of a process, while termination does not perform a transition, the actual re-initialization never takes place. Nevertheless, before the termination takes place, it is evaluated whether the re-initialization has a possible solution, which is reflected in the use of the satisfiability operator $d^?$ in some of the axioms. This peculiar behavior for termination, is visible in one of the distribution axioms for sequential composition.

$[true] \gg x \approx x$	$d \gg (x \oplus y) \approx d \gg x \oplus d \gg y$
$\lfloor false \rfloor \gg x \approx \delta$	$(d \gg a) \odot x \approx d \gg a \odot x$
$d \gg \delta \approx \delta$	$(d \gg c) \odot x \approx d \gg c$
$d \gg d' \gg x \approx (d \sim d') \gg x$	$(d \gg \epsilon) \odot x \approx d^? \gg x$
$d \gg x \oplus d' \gg x \approx (d \vee d') \gg x$	$(d \gg x) \rhd y \approx d \gg x \rhd y$
$c_{jmp} \gg c \approx c$	$(d \gg x) \lfloor\!\lfloor y \approx d \gg x \lfloor\!\lfloor y$
	$\partial_H(d \gg x) \approx d \gg \partial_H(x)$

Using reasoning on re-initialization clauses, we find that $([true] \vee [true]) \gg x \approx [true] \gg x$. A trivial consequence of this, is for example the equality $x \oplus x \approx x$, which was stated before as an axiom from basic process algebra, but can also be derived from the axioms on re-initialization and alternative composition. Again, using reasoning on re-initialization clauses, we find that $[true] \gg x \approx [true]^? \gg x$, and we may derive another axiom from basic process algebra: $\epsilon \odot x \approx x$.

As mentioned before, re-initialization does not distribute over forced-synchronization. Because of this, many of the axioms given before for forced-synchronization have to be repeated in the light of re-initializations. In some of these axioms, termination plays its peculiar role again.

$$
\begin{array}{rcll}
d \gg \epsilon \mid d' \gg \epsilon & \approx & (d^? \wedge d'^?) \gg \epsilon & \\
d \gg \epsilon \mid d' \gg a \odot x & \approx & \delta & \\
d \gg a \odot x \mid d' \gg a' \odot y & \approx & (d \wedge d') \gg (a \gamma a') \odot (x \parallel y) & \text{if } a \gamma a' \text{ defined} \\
d \gg a \odot x \mid d' \gg a' \odot y & \approx & \delta & \text{if } a \gamma a' \text{ undefined} \\
d \gg \epsilon \mid d' \gg c \rhd x & \approx & (d^? \sim d') \gg c \rhd x & \\
d \gg c \rhd x \mid d' \gg a \odot y & \approx & \delta & \\
d \gg c \rhd x \mid d' \gg c' \rhd y & \approx & ((d \sim c_{jmp}) \wedge (d' \sim c'_{jmp})) \gg & \\
 & & (c \wedge c') \rhd \begin{pmatrix} x \between c' \blacktriangleright y \oplus \\ y \between c \blacktriangleright x \oplus \\ x \mid c' \blacktriangleright y \oplus \\ y \mid c \blacktriangleright x \end{pmatrix} &
\end{array}
$$

Termination only takes place, if both re-initializations are satisfiable, independent of each other. If synchronizing actions are re-initialized, both re-initializations should be satisfied, i.e. both processes should agree on the change of valuation. In particular, if $a \gamma a' = a''$, and a is re-initialized by an assignment $x^+ = x^- + 1$, we find $\left[x \mid x^+ = x^- + 1\right] \gg a \mid a' \approx \left[x \mid x^+ = x^- + 1\right] \gg a \mid [\mathit{true}] \gg a' \approx \left(\left[x \mid x^+ = x^- + 1\right] \wedge [\mathit{true}]\right) \gg (a \gamma a') \approx \left[\mathit{false}\right] \gg (a \gamma a') \approx \delta$. The action a' does not allow any changes in the variables. In the calculation, this is reflected in the fact that $[\mathit{true}]$ does not allow any changes in valuations. A deadlock is the result of this disagreement between the re-initializations of a and a'.

Re-initialization shows a clear distinction between the way in which termination behaves in parallel to actions and in parallel to flows. Since actions cannot synchronize with termination, we find that termination (with a possible re-initialization) is delayed, i.e. $d \gg a \parallel d' \gg \epsilon \approx d \gg a \odot d'^? \gg \epsilon$, while termination must take place before a flow is executed, hence $d \gg c \parallel d' \gg \epsilon \approx d'^? \gg d \gg c$. The axiom in which flows synchronize after re-initialization, is quite complicated due to our decision to make it possible for flow clauses to perform spontaneous re-initializations. When synchronizing, these flow clause re-initializations should be taken into account. If we restrict ourselves to flow clauses in which all variables are continuous, and are not allowed to jump (as is done in hybrid automata for example), i.e. clauses of the form $\left\{ \mathcal{V}_{\mathrm{m}} \mid P_{\mathrm{f}} \right\}$, we find the equality $d \gg c \rhd x \mid d' \gg c' \rhd y \approx (d \wedge d') \gg (c \wedge c') \rhd \big(x \between c' \blacktriangleright y \oplus y \between c \blacktriangleright x \oplus x \mid c' \blacktriangleright y \oplus y \mid c \blacktriangleright x\big)$, which is more in line with the intuition that both re-initialization clauses and flow clauses are synchronized. The proof of equality relies on the observation that, in case of continuity, $c_{jmp} = c^?_{jmp}$ (no jumps, hence only satisfiability) and $(d_0 \sim d_1^?) \wedge (d_0' \sim d_1'^?) = (d_0 \wedge d_0') \sim (d_1^? \wedge d_1'^?)$.

Derivation rule (9) in Table 6, expresses how a process re-initialization can restrict the choice for the first transition of a flow clause. A useful application of this rule is in recognizing a particular solution of a differential equation given a certain initial condition. For example, consider the flow clause $\left\{ x, t \mid \dot{x} = x \wedge \dot{t} = 1 \right\}$. Clearly, $x = e^t$ is a solution of the differential equation $\dot{x} = x$, and if initially $t = 0$ and $x = 1$, this solution is unique. Using derivation rule (9), we now find the following equivalence:

$$
\begin{array}{l}
\left[\begin{array}{c|c} x & x^+ = 1 \\ t & t^+ = 0 \end{array}\right] \gg \left(\begin{array}{c|c} x & \dot{x} = x \\ t & \dot{t} = 1 \end{array}\right) \\
\approx \\
\left[\begin{array}{c|c} x & x^+ = 1 \\ t & t^+ = 0 \end{array}\right] \gg \left(\begin{array}{c|c} x & x = e^t \\ t & \dot{t} = 1 \end{array}\right) \rhd \left(\begin{array}{c|c} x & \dot{x} = x \\ t & \dot{t} = 1 \end{array}\right).
\end{array}
$$

Note, that t and x are both not allowed to jump. Otherwise, the flow clauses in this example might execute undesired re-initializations. Derivation rule (9), also expresses the repetitive character of flow clauses. This is illustrated using $d = [true]$ and $c' = c$. We then find the equivalence $c \approx c \rhd c$.

Derivation rule (10), also expresses this repetitive character. This is illustrated by taking $c = c' = c''$, we then find again $c \approx c \rhd c$. Furthermore, derivation rule (10) expresses that if we can divide a flow clause c into two (possibly overlapping) clauses c' and c'', then the first transition taken by c can be mimicked by either c' or c''. An application of this rule, is that a solution of a flow clause can be split off even if there is no re-initialization. For example, the flow clause $\left[\dot{x} = 3x^{\frac{2}{3}} \wedge \dot{t} = 1\right]$ contains a set of differential equations with solutions $x = 0$ and $x = t^3$, if initially $x = 0$ and $t = 0$. However, for other initial conditions, other solutions are possible. Using derivation rule (10), we find the following equality, which describes exactly that $x = 0$ and $x = t^3$ are two possible trajectories of this flow clause:

$$
\begin{aligned}
& \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right] \\
\approx\ & \left(\left[\begin{matrix} x = 0 \\ \dot{t} = 1 \end{matrix}\right] \oplus \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right]\right) \rhd \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right] \\
\approx\ & \left(\left[\begin{matrix} x = 0 \\ \dot{t} = 1 \end{matrix}\right] \oplus \left(\left[\begin{matrix} x = t^3 \\ \dot{t} = 1 \end{matrix}\right] \oplus \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right]\right) \rhd \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right]\right) \\
\rhd\ & \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right] \\
\approx\ & \left[\begin{matrix} x = 0 \\ \dot{t} = 1 \end{matrix}\right] \rhd \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right] \\
\oplus\ & \left(\left[\begin{matrix} x = t^3 \\ \dot{t} = 1 \end{matrix}\right] \rhd \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right]\right) \rhd \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right] \\
\oplus\ & \left(\left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right] \rhd \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right]\right) \rhd \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right] \\
\approx\ & \left[\begin{matrix} x = 0 \\ \dot{t} = 1 \end{matrix}\right] \rhd \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right] \oplus \left[\begin{matrix} x = t^3 \\ \dot{t} = 1 \end{matrix}\right] \rhd \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right] \oplus \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right] \\
\approx\ & \left(\left[\begin{matrix} x = 0 \\ \dot{t} = 1 \end{matrix}\right] \oplus \left[\begin{matrix} x = t^3 \\ \dot{t} = 1 \end{matrix}\right]\right) \blacktriangleright \left[\begin{matrix} \dot{x} = 3x^{\frac{2}{3}} \\ \dot{t} = 1 \end{matrix}\right].
\end{aligned}
$$

Note that, in contrast to the example for derivation rule (9), we do not need to require that x and t do not jump.

3.2. Congruence and soundness

Rests us to show, that robust bisimilarity is a congruence for all the operators of HyPA, and that all the derivations that can be made about process terms, indeed lead to sound statements about the robust bisimilarity of these terms. In other words, we need to prove the following theorems.

Theorem 8 (Congruence). *Robust bisimilarity is a congruence for all operators of HyPA.*

Proof. In Appendix A, we sketch the proof of this theorem, by giving witnessing robust bisimulation relations. The proof for parallel composition is worked out in detail, since it relies on the notion of robustness against interference. □

Theorem 9 (Soundness). *If, for two process terms p and q, we find HyPA $\vdash_E p \approx q$ then $p \underline{\leftrightarrow} q$.*

Proof. As mentioned before, robust bisimilarity coincides with the notion of bisimilarity used in [46]. Hence, the result shown in Appendix A of that report, that every derivation in HyPA is sound for bisimilarity, transfers to robust bisimilarity. In Appendix C of this paper, we give a summary of that proof, adapted for robust bisimilarity. We also give a witness relation for soundness of the axiom $(x \odot \delta \rhd y) \odot z \approx x \odot \delta \rhd y \odot z$, which was not in [46]. □

3.3. Recursion principles

When reasoning about recursion, it is often useful to have a principle that claims that a solution of certain recursive specifications exists and is unique. That a solution exists follows directly from the operational semantics of HyPA, but it is not always clear that particular solution is the only process satisfying the recursive equations. Let us first define what we mean by solution.

Definition 10 (*Solution*). Let E be a recursive specification. An interpretation $S \in \mathcal{V}_r \to \mathcal{T}(\mathcal{V}_r)$ of recursion variables as process terms, is a *solution* of E (denoted $S \models E$) if for every recursive definition $X \approx p \in E$ we have $S(X) \underline{\leftrightarrow} S(p)$, where $S(p)$ denotes the process term induced by application of S to the variables of p. In particular, $S(X)$ is called a solution of $X \approx p \in E$.

The *recursive specification principle* RSP, which is quite standard in process algebra [49], states that so called guarded recursive specifications have at most one solution. For HyPA, guardedness of a recursive specification is defined as follows.

Definition 11 (*Guardedness*). An open process term p is *guarded* if all occurrences of process variables in p, are in the scope of an action prefix $a \odot \_$ or a flow prefix $c \rhd \_$. A recursive specification E is *guarded* if for each recursive definition $X \approx p \in E$, p can be rewritten into a guarded process term using the axiomatization of HyPA.

This leads to the principle given in Table 7.

Table 7
Recursive specification principle

$\dfrac{S \models E,\ S' \models E,\ E \text{ guarded}}{S(X) \approx S'(X)}\ X \in \mathcal{V}_r$

Theorem 12. *The recursive specification principle is sound.*

Proof. This is proven in Appendix D. □

As an example, the process terms $\epsilon \oplus a \odot d \gg X$ and $c \rhd (X \oplus Y)$ are guarded, while the process terms $c \blacktriangleright X$ and $X \oplus a \odot X$ are not. That unguarded recursive equations do not necessarily have a unique solution, can be seen from the fact that the processes c and $\lfloor true \rfloor$ are both solutions of the equation $Y \approx c \blacktriangleright Y$, and also the equation $Z \approx Z \oplus a \odot Z$ has multiple solutions, some of which even execute flow transitions! From RSP, it follows that the recursive specification $X_1 \approx \epsilon \oplus a \odot d \gg X_2, X_2 \approx c \rhd (X_1 \oplus X_2)$ has unique solutions for X_1 and X_2.

Indeed, the fact that the disrupt operator is unguarded, while it occurs naturally in many models of hybrid systems, implies that some extra care needs to be taken during the modeling stage in order to ensure that calculation remains possible. For example, the Boiler process of Section 2.4 may seem unguarded at first sight, but reasoning about the re-initialization clauses will show that the disrupt operator may be replaced by a left-disrupt, which makes the process guarded. In the model of the colliding masses, left-disrupt was used especially to guarantee guardedness of the definitions.

Another possible approach, that is not discussed in this paper, is to consider only the solution that is defined by the operational semantics of HyPA. The solution of $X \approx X$, for example, would be deadlock δ, while the solution of $Z \approx c \blacktriangleright Z$ is c. Also, the left-disrupts in the definitions of the colliding masses could be replaced by disrupts. The solution of the operational semantics is the same for both. Calculation with this view on recursion, however, is often more elaborate.

3.4. Conservativity and rewriting

One of the things that can be concluded about HyPA, using the given axiomatization, is that it is a conservative extension of the process algebra ACP [7]. This illustrates that HyPA does not violate the general ideas behind this process algebra.

Theorem 13 (Conservativity). *HyPA is a conservative extension of ACP* (*except for notational differences* $\oplus$ *and* $\odot$), *meaning that for every two closed ACP terms* p *and* q, *we find that ACP* $\vdash p \approx q$ *if and only if HyPA* $\vdash p \approx q$.

Proof. One direction of the proof, that derivations in ACP can be mimicked in HyPA, is based on the fact that all axioms of ACP can be derived in HyPA. The other direction relies on the construction of a relation that shows that if two closed ACP terms have robustly bisimilar semantics in HyPA, then they have bisimilar semantics in ACP. Completeness of ACP for bisimilarity then leads to the conclusion that derivably equal processes in HyPA also have a derivation showing equality in ACP. The complete proof of this claim can be found in [46]. □

Furthermore, like in ACP, it is possible to define a set of basic terms into which every closed term can be rewritten. These basic terms clearly show that the parallel compositions can be eliminated from all closed terms.

Definition 14 (*Basic terms*). A *basic term* is a closed term of the following form:

$$N ::= d \gg \epsilon \mid d \gg a \odot N \mid d \gg c \rhd N \mid N \oplus N,$$

where $a \in \mathcal{A}$, $c \in C$, and $d \in D$.

Theorem 15 (Elimination). *Every closed term is derivably equal to a basic term.*

Proof. In Appendix E, a strongly normalizing rewrite system is given that achieves this, based (in principle) on reading the axioms as rewrite rules from left to right, modulo the use of unit elements. □

We conjecture that this elimination result can be extended to a linearization result, meaning that we expect to be able to rewrite a broad class of guarded recursive specifications of a HyPA process into a linear form in which we only use recursion over basic terms.

The usefulness of elimination of the parallel composition, was already noted in the introduction. It was pointed out there, that the notion of robust bisimilarity we use is very strong, because all possible valuations of the variables are taken into account at every point in time. Many weaker notions of equivalence, while still preserving interesting analysis properties, are not sensitive to the valuation of variables. Those equivalences, often, are not congruent for the parallel composition operator. Therefore, algebraic reasoning about those notions in the context of parallel composition becomes difficult. This is a known phenomenon in process theory, and it is caused by the possibility of interference in the value of shared variables (see for example [50]). Many different solutions have been proposed, also in the field of hybrid systems. For example, in the hybrid automaton theory of [17], the authors propose a restriction (called compatibility of automata) on the systems that may be placed in parallel, to ensure that no interference occurs. This is a perfectly reasonable way of handling the problem, but it has the disadvantage that we have to add extra variables, if we want to model processes that intentionally interfere, like the control system shown in the introduction.

HyPA is, in principle, focussed on being general. We start out by using a very general parallel composition, that is defined for all possible processes, and necessarily end up with an equivalence that is very strong, but is at least a congruence for this composition. Now, the elimination result allows us to eliminate the parallel composition from the process description. After elimination, we can start to use algebraic reasoning on a weaker notion of equivalence to analyze the specific properties we are interested in. This method may turn out to be less practical than the road followed by [17], because the elimination of parallel compositions can become quite cumbersome. On the other hand, it may also be possible to formulate derivation rules for reasoning about weaker notions of equivalence, that express a kind of conditional congruence 'under compatibility'. In this way, other methods can be imported into HyPA.

As an example of rewriting into basic terms, we can rewrite the steam boiler system of the previous section into the following description, in which parallel composition and encapsulation are eliminated:

$$\begin{aligned}
&\text{Boiler} \approx \text{Open} \oplus \text{Closed},\\
&\text{Open} \approx d_0 \gg c_o \rhd (d_1 \gg cl \odot \text{Closed} \oplus d_2 \gg \text{Open}),\\
&\text{Closed} \approx d_0 \gg c_c \rhd (d_2 \gg \text{Closed} \oplus d_3 \gg op \odot \text{Open}),
\end{aligned}$$

with

$$
\begin{aligned}
d_0 &\equiv \left[t \,\middle|\, t^+ = 0\right],\\
d_1 &\equiv \left[t^- = T\right] \wedge \left[w^- \geqslant w_{max} - w_{safe}\right],\\
d_2 &\equiv \left[t^- = T\right] \wedge \left[w_{min} + w_{safe} \leqslant w^- \leqslant w_{max} - w_{safe}\right],\\
d_3 &\equiv \left[t^- = T\right] \wedge \left[w^- \leqslant w_{min} + w_{safe}\right],\\
c_o &\equiv \left\{ t, w \;\middle|\; \dot{t} = 1 \wedge t \leqslant T \wedge \dot{w} = v - s \wedge s_{min} \leqslant s \leqslant s_{max} \wedge v = v_{in} \right\},\\
c_c &\equiv \left\{ t, w \;\middle|\; \dot{t} = 1 \wedge t \leqslant T \wedge \dot{w} = v - s \wedge s_{min} \leqslant s \leqslant s_{max} \wedge v = 0 \right\}.
\end{aligned}
$$

Notice that this rewriting is done here over a recursive definition, hence is an example of linearization of such process descriptions. Looking at the axiomatization, one might expect that $d_0, \ldots, d_3$ would contain clauses of the form c_{jmp}, but those (and other distracting terms) are eliminated using calculation on re-initialization clauses. Furthermore, looking at the original recursive definition, one might suspect that it is non-guarded, but again, calculation on the re-initialization clauses shows that the definition can be rewritten into a guarded one. Performing the actual elimination by hand is very cumbersome, and leads to a very long calculation, which we left out for reasons of space. Currently, tools are being developed for (partial) automation of such calculations. Using a preliminary version of one such tool, a mistake in the original calculation on the steam boiler was found already. Hence the difference between the result presented here and in [46].

One result that is missing, so far, is a proof that the given axiomatization is complete for robust bisimilarity of closed terms. I.e. a proof that for closed terms p and q, if $p \leftrightarrows q$ then also HyPA $\vdash p \approx q$. We do not exclude the possibility yet, modulo completeness of the logical equivalence of flow predicates and re-initialization predicates, but the fact that the number of flows that are a solution of a flow clause, and the number of valuation jumps that are a solution of a re-initialization clause may be infinite, complicates matters seriously.

4. Related work

In this section, we compare HyPA, in an informal way, to hybrid formalisms that were previously developed.

4.1. Hybrid automata

One of the most influential of all hybrid formalisms, is the hybrid automaton formalism described by Henzinger [16]. These automata consist of nodes in which certain differential equations are active under an invariant, and of guarded transitions between those nodes that model discrete actions. For example, the steam boiler example could be modeled as the hybrid automaton depicted in Fig. 8.

In the formal definitions of [16], a discrete action is associated with each and every transition. Note, however, that in the same paper there are several examples of hybrid automata with transitions without an associated action. We assume that this means that implicitly there is there is some special action, say τ, that does not have to synchronize with other events in case of parallel composition. The fact that, in HyPA, it is not necessary to add intermediate actions in order to switch between continuous behaviors, is one of the

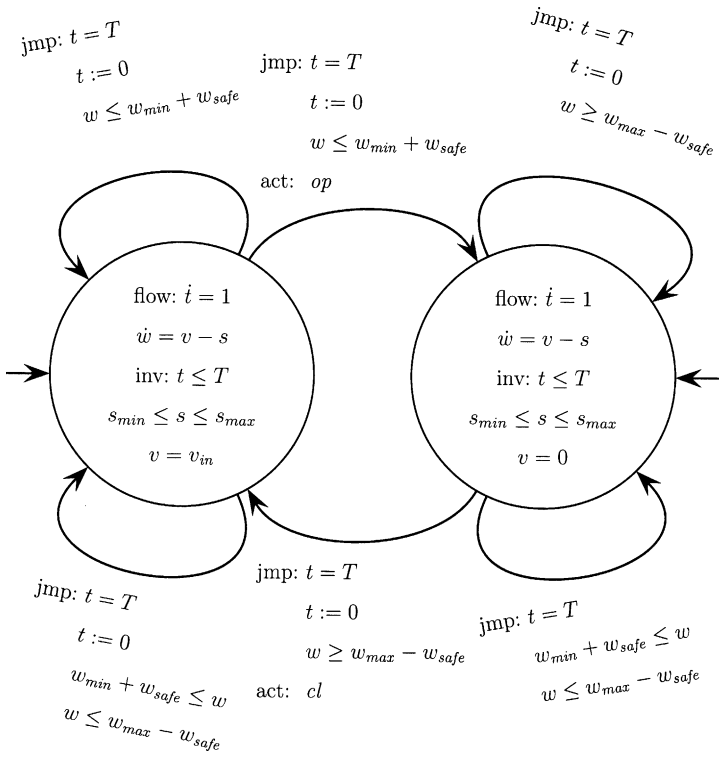

Fig. 8. Example of a hybrid automaton modelling a steam boiler.

reasons why we believe that a translation of HyPA into hybrid automata is impossible in general.

A translation of hybrid automata into HyPA, however, seems to be possible. A (part of a) general hybrid automaton is depicted in Fig. 9. Such an automaton is easily translated into a hybrid process algebraic term, using the following observations:

- The flow predicate P_f in a node of an automaton, describes flows in a similar way as in HyPA. Only, in hybrid automata, all flows are continuous. Hence, we take $V = \mathcal{V}_{\mathrm{m}}$ and find the clause $\left\{ \mathcal{V}_{\mathrm{m}} | \, P_f \right\}$. Furthermore, since hybrid automata only allow differentiable solutions of flow predicates, we adopt that notion of solution for our flow predicates as well.
- The invariant P_i in a node, is a predicate that can be used in a flow clause, but can also be transformed to be used in a re-initialization clause, since only variables from the set $\mathcal{V}_{\mathrm{m}}$ are used in it. The semantics of hybrid automata, contain a kind of look-ahead such that after a transition to a certain node, the invariant of that node must hold. Otherwise the transition cannot be taken. Translating this to HyPA means that in re-initializations the predicate P_i^+, of the next node, should hold. Recall that we have defined P^+ in Section 2.2, as a transformation of a predicate P on $\mathcal{V}_{\mathrm{m}}$ in which every variable x is replaced by x^+.
- The transitions of hybrid automata contain actions a. In translation, those actions disrupt the flow clauses. Furthermore, the jump condition P_j on a transition is translated into a re-initialization that acts on these actions. Again, we take $V = \mathcal{V}_{\mathrm{m}}$, and assume

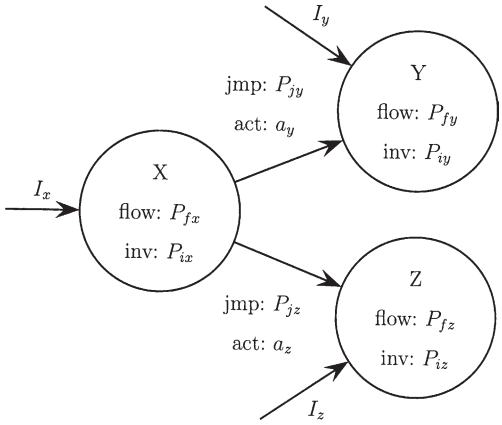

Fig. 9. General example of a hybrid automaton.

that it is specified in the jump condition which variables may change, and which remain constant.

- In a hybrid automaton the initial states are indicated by the initial conditions. For each node X such an initial condition is given by means of a predicate I_x over the model variables.

Using these observations, the more general automaton in Fig. 9 is translated into

$$X \approx \left\{ \mathcal{V}_{\mathrm{m}} \mid P_{fx} \wedge P_{ix} \right\} \blacktriangleright \begin{pmatrix} \left[\mathcal{V}_{\mathrm{m}} \middle| P_{jy} \wedge P_{iy}^{+} \right] \gg a_y \odot Y \\ \oplus \\ \left[\mathcal{V}_{\mathrm{m}} \middle| P_{jz} \wedge P_{iz}^{+} \right] \gg a_z \odot Z \end{pmatrix},$$

$$HA \approx \left[I_x^{+} \right] \gg X \oplus \left[I_y^{+} \right] \gg Y \oplus \left[I_z^{+} \right] \gg Z.$$

Of course, this is not a formal translation. The semantics of hybrid automata as given in [16] is one of timed transition systems, while the hybrid transition systems we use here are subtly different. We conjecture that it is possible to transform the flow transitions of the hybrid transition system into timed transitions, and the action transitions of the hybrid transition system into action transitions of a timed transition system, by abstracting away from all valuations. However, this is left as a subject for future research. The comparison with hybrid automata is merely intended to give an intuition on how the existing hybrid theories fit into our hybrid process algebraic framework.

4.2. Other process algebras

With respect to process algebras for hybrid systems, there are four related works that we must consider. One, hybrid CSP, was already introduced in 1994 by Jifeng [14]. The others, ϕ-calculus [15], hybrid χ [11], and $\mathrm{ACP}_{\mathrm{hs}}^{\mathrm{srt}}$ [13], are very recently introduced.

Hybrid CSP has a semantics in which each process represents a set of hybrid traces. Such a hybrid trace, consists of a function of a continuous closed time domain to valuations, a function of that same domain to sequences (that gives the empty sequence except for on a finite set of time-points), and a few predicates (like termination). A system is then modeled in hybrid CSP, by giving a predicate that defines which traces are in the system. Comparable to the way that HyPA has atomic processes and operators, hybrid CSP has atomic predicates, and predicate operators. Apart from the fact that a trace semantics does

not respect branching properties of a system, hybrid CSP also has the drawback that in parallel composition the continuous variables of the composed systems are assumed to be disjoint, and that assignments can only be made to programming variables, and not to continuous variables. We suspect, however, that these problems can be solved by defining new predicate operators, and that the author of [14] did not see the need for them at the time. Interestingly, there are operators defined in [14] whose function is not easily translated into HyPA. The main reason for this, is that clocks need to be modeled explicitly in HyPA, while they are often a functional part of the operators of hybrid CSP. Again, we conjecture, that HyPA can be extended with operators that mimic those of hybrid CSP, should the need arise.

The ϕ-calculus has a semantics based on timed transition systems, and given this, has a very interesting way of dealing with parallelism. As we already mentioned in the introduction, ϕ-calculus regards continuous behavior to be a property of the environment, rather than a property of the ϕ-calculus program. Execution starts with an empty environment and, while running the program, differential equations (or rather their vector-field equivalents) and invariants, are added and replaced, by (in an interleaving manner) executing so-called environmental actions. The upshot of this, is that it is not necessary to require that parallel programs have distinct continuous variables, but still, the semantics of the parallel composition of ϕ-calculus does not coincide with our intuition that continuous behavior should simply satisfy both processes. Furthermore, because a vector-field is used as a representation of differential equations in the environment, ϕ-calculus can only handle differential equations with unique solutions (hence, not for example the equation $\dot{x} = 3x^{\frac{2}{3}}$). Also, the notion of equivalence that arises from using bisimilarity in combination with environmental actions, makes that only syntactically equal differential equations are actually considered equal. This is a drawback that might be solved by some kind of abstraction, but it still has an artificial feel to it. Comparing ϕ-calculus to HyPA, we may conclude that, due to (amongst others) the environmental action approach, not all HyPA processes can be translated into ϕ-calculus. Conversely, the fact that the environmental actions of ϕ-calculus have a maximal progress semantics, ϕ-calculus programs cannot be translated into HyPA. This, however, can be solved by extending HyPA with an urgency operator, as was done for χ and hybrid χ in [5,11].

As we mentioned already in the introduction, HyPA is developed in close cooperation with the researchers developing hybrid χ. Research on the language hybrid χ, as a modeling and simulation language for process control, started in 1982 [51], and has since been through many stages of development, including an extension with hybrid description constructs. In 2002 [5], a formal operational semantics, based on CSP rather than ACP, was defined for the discrete-time part of the language, and recently, a formal semantics has been given for the hybrid part as well [11]. It is interesting to see that many of the theoretical aspects of HyPA (like the use of hybrid transition systems), have been applied in the formal semantics of hybrid χ, while on the other hand, the future extensions of HyPA are very likely to be inspired by the modeling strengths of hybrid χ, including their abstraction operators and possibly the maximal progress operator. As research progressed, both languages seem to have evolved more and more towards each other, and it is not unthinkable that these paths ultimately converge.

In [13] a combination of the process algebra with continuous relative timing of [45] and the process algebra with propositional flows of [52], lead to a (only subtly) different algebra, that is also suited for the description of hybrid systems. The development of

this algebra and of HyPA has been largely independent, and it is surprising to see how many similarities exist between the two. Nevertheless, due to different starting points and intuitions, also some differences can be found.

The process algebra of Bergstra and Middelburg [13], was intended to be a conservative extension of timed ACP, while HyPA was intended to be an extension of 'normal' ACP. This gives rise to the most important difference, in our opinion, between the two languages, which is that in [13], a time-deterministic setting was chosen (as it was discussed in Section 2.2), while for HyPA time-non-determinism is assumed (which is more in line with the hybrid automaton approach [16]). As a matter of fact, in hybrid χ, two choice operators exist, one for each view on time. Another difference is that in [13], there was the intension to give an algebraic theory of hybrid automata, which leads to the modeling choice that switching between continuous behaviors can only take place through the use of discrete actions, while in HyPA switching can be arbitrary. This is illustrated, by the fact that the passing of time during which physical behavior takes place, is modeled explicitly in [13], while, for HyPA, time passing is implicit when writing down a flow clause.

4.3. Control theory formalisms

The formalisms used in control theory to describe hybrid systems can, from a HyPA point of view, be classified into two kinds. The first kind, are formalisms regarding continuous time behavior, while the second kind regards time to evolve discretely. Roughly speaking, continuous time models can be translated into HyPA using flow clauses, while the discrete models can (amongst many other possibilities that we do not show here) be translated into re-initialization clauses, acting on a 'time-step' process. Computational actions and sequential compositions of processes, only occasionally play a role in control theory. Mode switching, on the other hand, is a central aspect. In this paragraph, we sketch the general translation of several control theory formalisms into HyPA. We do not intend to be complete, but rather want to give a feel for the relation between HyPA and control theory. Furthermore, one has to keep in mind that control theory usually reasons about trace equivalence of systems, while HyPA is primarily concerned with (robust) bisimilarity.

With respect to the continuous time models, we conjecture that most of them can be translated into either one single flow clause c or, in more complicated cases, into one single recursive term of the form

$$\mathrm{CT} \approx (c_0 \oplus \cdots \oplus c_n) \rhd \mathrm{CT},$$

where $c_0 \cdots c_n$, denote clauses representing the different continuous modes a system can be in. If a system can be modeled using only three continuous variables, namely the state variable $x \in \mathbb{R}^l$, the output variable $y \in \mathbb{R}^m$ and the input variable $u \in \mathbb{R}^n$, and using only clauses of the form

$$c_i = \left\{ x \;\middle|\; \begin{array}{l} \dot{x} = A_i x + B_i u + f_i \\ y = C_i x + D_i u + g_i \\ (x, u) \in H_i \end{array} \right\},$$

with A_i, B_i, C_i and D_i matrices of appropriate dimensions, and H_i a convex polyhedron (i.e. constructed from a finite set of inequalities), for every i, then we say that CT is a *continuous time piecewise affine* system [31]. If a system can be modeled as one single continuous flow clause, using the variables $v, w \in \mathbb{R}^s$ in addition to x, y and u, and if this flow clause is of the form

$$c = \left(x \;\middle|\; \begin{array}{l} \dot{x} = Ax + B_1u + B_2w \\ y = Cx + D_1u + D_2w \\ v = E_1x + E_2u + E_3w + e_4 \\ 0 \leqslant v \perp w \geqslant 0 \end{array} \right),$$

then we say that the system is a *continuous time linear complementarity* system [30]. Here, A, B_1, B_2, C, D_1, D_2, E_1, E_2 and E_3 are matrices of appropriate dimensions, e_4 is a constant vector and $0 \leqslant v \perp w \geqslant 0$ denotes that the vectors v and w are orthogonal (i.e. $0 \leqslant v$, $0 \leqslant w$ and $v^{\mathrm{T}}w = 0$).

Discrete time models can be translated into the HyPA term

$$\mathrm{DT} \approx (d_0 \vee \cdots \vee d_n) \gg \mathrm{Timestep} \odot \mathrm{DT},$$

with

$$\mathrm{Timestep} \approx \left[t \,\middle|\, t^+ = 0\right] \gg \left(\{t\} \cup V \;\middle|\; \begin{array}{l} \dot{t} = 1 \\ t \leqslant T_s \\ \bigwedge\limits_{j \in J} \dot{x_j} = 0 \end{array} \right) \blacktriangleright \left[t^- = T_s\right] \gg \epsilon.$$

Here, the set $V = \{x_j \mid j \in J\}$ denotes the set of all variables that are used in the re-initialization clauses $d_0 \cdots d_n$, describing the discontinuous changes over time. Timestep denotes the progress of time with one sample time $T_s > 0$, during which the variables x_j are supposed to remain constant. Similar to the continuous case, if (and only if) $V = \{x, y, u\}$, and for all re-initializations (with $i \in [0, n]$) we find

$$d_i = \left[x, y, u \;\middle|\; \begin{array}{l} x^+ = A_ix^- + B_iu^- + f_i \\ y^+ = C_ix^+ + D_iu^+ + g_i \\ y^- = C_ix^- + D_iu^- + g_i \\ (x^+, u^+) \in H_i \\ (x^-, u^-) \in H_i \end{array} \right],$$

with A_i, B_i, C_i and D_i matrices of appropriate dimensions, and H_i a convex polyhedron, we say that DT is a *discrete time piecewise affine* system [31]. Analogously, if (and only if) a system can be written in the form

$$DT = \left[\begin{array}{l} x, y \\ u, v, w \end{array} \;\middle|\; \begin{array}{l} x^+ = Ax^- + B_1u^- + B_2w^- \\ y^+ = Cx^+ + D_1u^+ + D_2w^+ \\ y^- = Cx^- + D_1u^- + D_2w^- \\ v^+ = E_1x^+ + E_2u^+ + E_3w^+ + e_4 \\ v^- = E_1x^- + E_2u^- + E_3w^- + e_4 \\ 0 \leqslant v^+ \perp w^+ \geqslant 0 \\ 0 \leqslant v^- \perp w^- \geqslant 0 \end{array} \right] \gg \mathrm{Timestep} \odot \mathrm{DT},$$

we say that it is a *discrete time linear complementarity* system [30].

A third type of discrete control formalism is *discrete time mixed logical dynamical* systems [28]. Similarly to linear complementarity systems, these systems can be described using only one re-initialization clause. This time, however, the clause also reasons about variables that take value in the domain $\{0, 1\}$. A mixed logical dynamical system may use variables $x \in \mathbb{R}^l$, $y \in \mathbb{R}^m$ and $u \in \mathbb{R}^n$, and in addition, the variables $z \in \mathbb{R}^r$ and $w \in \{0, 1\}^s$, and can be written in the form

$$DT = \left[\begin{array}{l|l} & x^+ = Ax^- + B_1u^- + B_2w^- + B_3z^- \\ & y^+ = Cx^+ + D_1u^+ + D_2w^+ + D_3z^+ \\ x, y & y^- = Cx^- + D_1u^- + D_2w^- + D_3z^- \\ u, v, w & E_1x^+ + E_2u^+ + E_3w^+ + E_4z^+ \leqslant e_5 \\ & E_1x^- + E_2u^- + E_3w^- + E_4z^- \leqslant e_5 \end{array}\right] \gg \text{Timestep} \odot \text{DT}.$$

In [29], the relation between the discrete control formalisms described above is further worked out, and it turns out that most of them are equivalent under certain, from a physical point of view very reasonable, assumptions.

There is also another natural way of dealing with discrete time models in HyPA, and that is by using a flow-clause parametrization with discrete time rather than continuous time. Simply assume that time consists of the natural numbers only. If there is no interaction to be modeled between the discrete time processes and continuous time processes, then this is a valid approach as well. However, if interaction is necessary, then one must know what happens in between the discrete steps. The approach we have shown above, is known in control as the *zero-order hold* approach. There are many other ways to model the behavior in between re-initializations, but that is a topic outside the scope of this paper.

As we mentioned in the beginning of this paragraph, HyPA is primarily concerned with the notion of robust bisimilarity. However, suppose we would adopt language equivalence, or even some weaker appropriate notion of equivalence. This would mean that we probably loose congruence of parallel composition, but it would also mean that we might be able to abstract away from a lot of computational behavior and rewrite certain HyPA processes into one of the above forms. Since a lot of control theory is developed for those forms, this might greatly improve the analysis possibilities of HyPA.

5. Conclusions and future work

In this paper, the syntax, semantics and axiomatization were presented, of a hybrid process algebraic theory called HyPA. This theory is aimed at the description and analysis of hybrid systems. HyPA is a conservative extension of the process algebra ACP [7], with a constant representing termination, a disrupt operator in the style of LOTOS [8], and clauses [2] for the description of continuous and discontinuous behavior of model variables. More precisely, the set of discrete actions (and the communication function) and the predicates used for describing flows and re-initializations (and the corresponding solution notions) are parameters of the theory. The reason for this is that they are often problem-specific. Using the axiomatization of HyPA, closed terms can be rewritten into basic terms, in which all parallel compositions are eliminated.

HyPA turns out to be different from most existing hybrid formalisms, in two major ways. It has a hybrid transition system semantics, for which it is not necessary to distinguish between state variables and external variables in differential equations. This allows for a general definition of parallel composition in the style of ACP, that also allows continuous interaction between all model variables. Furthermore, discontinuities in the variables of differential equations do not need to be explicitly modeled by assignment actions. Alternatively, in HyPA it is explicitly written down when a variable is continuous. Apparent drawbacks of HyPA are its strong notion of equivalence, and the sometimes complex axiomatization. However, we have sketched, how by assuming the same properties that are common on hybrid automata (compatibility of parallel composed systems, and continuity

of all model variables), both the equivalence may be weakened, and the axiomatization becomes simpler. HyPA is very similar to the languages hybrid χ [11] and the hybrid process algebra of [13]. The differences are mainly found in the way time-determinism is treated, and in the way in which the passing of time is modeled implicitly or explicitly.

Future work on HyPA can be divided into five categories, given in arbitrary order:

- The first category, is a formalization of Section 4, comparing HyPA to other (hybrid) formalisms. Clearly, since hybrid χ and the works of [13] are very similar, a formal comparison is indispensable. Also, formal comparisons with hybrid automata, ϕ-calculus, and hybrid Petri nets, are important. Translations to and from those formalisms are useful, in order to be able to use analysis techniques from one, in the other formalism. This, of course, is also the case for various control formalisms and techniques.
- The second category, is the application of HyPA to a number of (larger) case studies. Only this reveals whether the way of modeling we have chosen is indeed as convenient as expected, and whether practical theorems can be formulated to support the analysis of hybrid systems.
- The third category encompasses work on showing that the axiomatization of HyPA, modulo calculation on clauses, is complete (or can be made complete) for the notion of robust bisimilarity. Also, extending the result for rewriting closed terms into basic terms, to rewriting of recursive specifications into a linear form, is essential for the analysis of systems.
- The fourth category of future work, is the extension of the theory with abstraction. Also, extension with system theoretic concepts like, for example, a metric or topology on the state-space [53], or other notions of limit behavior [54], may then come into play. One of the classical problems in the hybrid systems field, namely the analysis of Zeno-behavior, where infinite sequences of actions converge to a certain point, arises from such a metric, and we feel that a truly hybrid semantical model should include it. It is important to note, that without abstraction, our current notion of equivalence is strong enough to capture Zeno-behavior, simply because process terms need to be equivalent for all valuations of variables, including Zeno-points. After abstraction of certain variables, however, Zeno-behavior of those variables cannot be distinguished anymore, and therefore a new notion of equivalence might be needed. Other types of abstraction, like abstraction from actions [7,43], would also greatly improve the analytic powers of HyPA. Also for those, new notions of (robust) bisimilarity, known in classical process algebra for example, branching bisimilarity, or observational equivalence, are needed.
- The fifth category, is tool support. Calculations, even on a simple example such as the steam boiler, quickly become very cumbersome, tedious and error prone. This is a serious problem when applying the theory to any system of interesting size. Using the result that processes can be rewritten into basic terms using a strongly terminating rewriting system, makes that developing a very basic tool for partially automating these calculations should not be difficult.

Acknowledgements

Finally, we would like to thank Paul van den Bosch, Bert van Beek, Jan Friso Groote, Maurice Heemels, Aleksandar Juloski, Ka Lok Man, Kees Middelburg, Mohammad

Mousavi, Ramon Schiffelers, Frits Vaandrager and Tim Willemse, for their comments during several stages of the development of this paper. We would like to thank Peter van den Brand for his quickly but thoroughly developed linearization tool that made calculations easier and more reliable for us.

Appendix A. Congruence

In this section, we prove that robust bisimilarity is a congruence for the operators of HyPA. For most of the operators, we only give the witnessing relations. For the parallel composition, we give the full proof. But before we present this proof, we need to pose a lemma that states that transitions that are labeled with a certain valuation, end in a state with that same valuation. This turns out to be vital in many of the proofs for congruence.

Lemma A.1 (Labelling). *A transition labeled with a valuation, leads to a state with that same valuation*:

- *If* $\langle x, \nu \rangle \overset{a,\nu'}{\mapsto} \langle y, \nu'' \rangle$ *then* $\nu' = \nu''$;
- *If* $\langle x, \nu \rangle \overset{\sigma}{\rightsquigarrow} \langle y, \nu' \rangle$ *and* $dom(\sigma) = [0, t]$ *then* $\nu' = \sigma(t)$.

Proof. This is obvious from the semantics of HyPA. It trivially holds for atomic processes, and all semantical rules of the operators of HyPA preserve this connection between labeling and state. □

Theorem A.2. *Robust bisimilarity* $\underline{\leftrightarrow}$ *is a congruence for all the operators of HyPA*.

Proof. We show the proof for parallel composition. For the other operators, we only give the witnessing relations.

Congruence means, that if $p \underline{\leftrightarrow} p'$ and $q \underline{\leftrightarrow} q'$, then also $p \parallel q \underline{\leftrightarrow} p' \parallel q'$. Let $\mathscr{R}$ be a relation witnessing $p \underline{\leftrightarrow} p'$, and let $\mathscr{S}$ be a relation witnessing $q \underline{\leftrightarrow} q'$. Then, we construct the following relation:

$$\mathscr{U}_{\parallel} = \big\{((x \parallel y, \nu), (x' \parallel y', \nu')) \mid x, x', y, y' \in \mathcal{T}(\mathcal{V}_{\mathrm{r}}),\ \nu, \nu' \in Val,\ (x, \nu)\mathscr{R}(x', \nu'),\ (y, \nu)\mathscr{S}(y', \nu')\big\} \cup \mathscr{R} \cup \mathscr{S},$$

and prove that it is a robust bisimulation relation, witnessing $x \parallel y \underline{\leftrightarrow} x' \parallel y'$.

That it is a witness relation is trivial. That it is a robust relation, is straightforward from the fact that $\mathscr{R}$ and $\mathscr{S}$ are robust. That it is a bisimulation relation follows from the cases below.

The only interesting case, is where $(x \parallel y, \nu)\mathscr{U}_{\parallel}(x' \parallel y', \nu')$ for some $x, y, x', y' \in \mathcal{T}(\mathcal{V}_{\mathrm{r}})$ and $\nu, \nu' \in Val$. Note, that by definition of $\mathscr{U}_{\parallel}$ we may use the assumption that $(x, \nu)\mathscr{R}(x', \nu')$ and $(y, \nu)\mathscr{S}(y', \nu')$. We find the following subcases:

(1) $\langle x \parallel y, \nu \rangle \checkmark$, for which we need the assumption
   (a) $\langle x, \nu \rangle \checkmark \wedge \langle y, \nu \rangle \checkmark$. Using the fact that $\mathscr{R}$ and $\mathscr{S}$ are bisimulation relations, we find $\langle x', \nu' \rangle \checkmark$ and $\langle y', \nu' \rangle \checkmark$ and may readily conclude $\langle x' \parallel y', \nu' \rangle \checkmark$.

(2) $\langle x' \parallel y', \nu' \rangle \checkmark$, similar to the previous case.

(3) $\langle x \parallel y, \nu \rangle \overset{l}{\rightarrow} \langle z, \mu \rangle$, for which we need one of the following assumptions:
   (a) $l \in \Sigma \wedge \langle x, \nu \rangle \overset{l}{\rightsquigarrow} \langle z, \mu \rangle \wedge \langle y, \nu \rangle \checkmark$

From the fact that $\mathscr{R}$ is a bisimulation relation, we conclude that there exist z' and μ' such that $\langle x', \nu' \rangle \overset{l}{\rightsquigarrow} \langle z', \mu' \rangle$ and $(z, \mu)\mathscr{R}(z', \mu')$. From the fact that $\mathscr{S}$ is a bisimulation relation, we know $\langle y', \nu' \rangle \checkmark$. Finally, we conclude that $\langle x' \parallel y', \nu' \rangle \overset{l}{\rightarrow} \langle z', \mu' \rangle$ and $(z, \mu)\mathscr{U}_{\parallel}(z', \mu')$ using the fact that $\mathscr{R} \subseteq \mathscr{U}_{\parallel}$.

(b) $l \in \Sigma \wedge \langle y, \nu \rangle \overset{l}{\rightsquigarrow} \langle z, \mu \rangle \wedge \langle x, \nu \rangle \checkmark$
Similar to the previous case.

(c) $\exists_{z_x, z_y}\, l \in \Sigma \wedge z \equiv z_x \parallel z_y \wedge \langle x, \nu \rangle \overset{l}{\rightsquigarrow} \langle z_x, \mu \rangle \wedge \langle y, \nu \rangle \overset{l}{\rightsquigarrow} \langle z_y, \mu \rangle$
From the fact that $\mathscr{R}$ and $\mathscr{S}$ are bisimulation relations, we conclude that there exist z'_x,z'_y, μ_x and μ_y such that $\langle x', \nu' \rangle \overset{l}{\rightsquigarrow} \langle z'_x, \mu_x \rangle$ and $\langle y', \nu' \rangle \overset{l}{\rightsquigarrow} \langle z'_y, \mu_y \rangle$, with $(z_x, \mu)\mathscr{R}(z'_x, \mu_x)$ and $(z_y, \mu)\mathscr{S}(z'_y, \mu_y)$. Using Lemma A.1 we find $\mu' = \mu_x = \mu_y$ and finally conclude that $\langle x' \parallel y', \nu' \rangle \overset{l}{\rightarrow} \langle z'_x \parallel z'_y, \mu' \rangle$ with $(z, \mu)\mathscr{U}_{\parallel}$ $(z'_x \parallel z'_y, \mu')$.

(d) $\exists_{z_x}\, l \in A \wedge z \equiv z_x \parallel y \wedge \langle x, \nu \rangle \overset{l}{\mapsto} \langle z_x, \mu \rangle$
From the fact that $\mathscr{R}$ is a bisimulation relation, we find that there exist z'_x and μ', with $\langle x', \nu' \rangle \overset{l}{\mapsto} \langle z'_x, \mu' \rangle$ and $(z_x, \mu)\mathscr{R}(z'_x, \mu')$. Using Lemma A.1, we conclude that $\mu = \mu'$, and we can construct an interference $\iota \in \mathit{Val} \to \mathit{Val}$ such that $\iota(\nu) = \iota(\nu') = \mu$. Because $\mathscr{S}$ is robust, we may conclude $(y, \iota(\nu))\mathscr{S}(y', \iota(\nu'))$, and we finally find $\langle x' \parallel y', \nu' \rangle \overset{l}{\rightarrow} \langle z'_x \parallel y', \mu' \rangle$ with $(z, \mu)\mathscr{U}_{\parallel}(z'_x \parallel y', \mu')$.

(e) $\exists_{z_y}\, l \in A \wedge z \equiv x \parallel z_y \wedge \langle y, \nu \rangle \overset{l}{\mapsto} \langle z_y, \mu \rangle$
Similar to the previous case.

(f) $\exists_{z_x, z_y}\, l \in A \wedge z \equiv z_x \parallel z_y \wedge l \equiv (a\gamma b, \vartheta) \wedge$
$\langle x, \nu \rangle \overset{a,\vartheta}{\mapsto} \langle z_x, \mu \rangle \wedge \langle y, \nu \rangle \overset{b,\vartheta}{\mapsto} \langle z_y, \mu \rangle$
From the fact that $\mathscr{R}$ and $\mathscr{S}$ are bisimulation relations, we conclude that there exist z'_x,z'_y, μ_x and μ_y such that $\langle x', \nu' \rangle \overset{a,\vartheta}{\mapsto} \langle z'_x, \mu_x \rangle$ and $\langle y', \nu' \rangle \overset{b,\vartheta}{\mapsto} \langle z'_y, \mu_y \rangle$, with $(z_x, \mu)\mathscr{R}(z'_x, \mu_x)$ and $(z_y, \mu)\mathscr{S}(z'_y, \mu_y)$. Using Lemma A.1 we find $\vartheta = \mu_x = \mu_y$ and finally conclude that $\langle x' \parallel y', \nu' \rangle \overset{a\gamma b,\mu}{\rightarrow} \langle z'_x \parallel z'_y, \vartheta \rangle$ with $(z, \mu)\mathscr{U}_{\parallel}$ $(z'_x \parallel z'_y, \vartheta)$.

(4) $\langle x' \parallel y', \nu \rangle \overset{l}{\rightarrow} \langle z', \mu \rangle$, similar to the previous case.

The following relations witness congruence for the other operators, given that $\mathscr{R}$ witnesses $p \underline{\leftrightarrow} p'$ and $\mathscr{S}$ witnesses $q \underline{\leftrightarrow} q'$ as before:

$$\mathscr{U}_{\oplus} = \big\{((x \oplus y, \nu), (x' \oplus y', \nu')) \mid x, x', y, y' \in \mathcal{T}(\mathcal{V}_{\mathrm{r}}),\ \nu, \nu' \in \mathit{Val},\ (x, \nu)\mathscr{R}(x', \nu'),\ (y, \nu)\mathscr{S}(y', \nu')\big\} \cup \mathscr{R} \cup \mathscr{S},$$

$$\mathscr{U}_{\odot} = \big\{((x \odot y, \nu), (x' \odot y', \nu')) \mid x, x', y, y' \in \mathcal{T}(\mathcal{V}_{\mathrm{r}}),\ \nu, \nu' \in \mathit{Val},\ (x, \nu)\mathscr{R}(x', \nu'),\ (y, \nu)\mathscr{S}(y', \nu')\big\} \cup \mathscr{S},$$

$$\mathscr{U}_{\blacktriangleright} = \big\{((x \blacktriangleright y, \nu), (x' \blacktriangleright y', \nu')) \mid x, x', y, y' \in \mathcal{T}(\mathcal{V}_{\mathrm{r}}),\ \nu, \nu' \in \mathit{Val},\ (x, \nu)\mathscr{R}(x', \nu'),\ (y, \nu)\mathscr{S}(y', \nu')\big\} \cup \mathscr{S},$$

$$\mathscr{U}_{\triangleright} = \big\{((x \triangleright y, \nu), (x' \triangleright y', \nu')) \mid x, x', y, y' \in \mathcal{T}(\mathcal{V}_{\mathrm{r}}),\ \nu, \nu' \in \mathit{Val},\ (x, \nu)\mathscr{R}(x', \nu'),\ (y, \nu)\mathscr{S}(y', \nu')\big\} \cup \mathscr{U}_{\blacktriangleright},$$

$$\mathscr{U}_{\lfloor\!\lfloor} = \big\{((x \lfloor\!\lfloor y, \nu), (x' \lfloor\!\lfloor y', \nu')) \mid x, x', y, y' \in \mathcal{T}(\mathcal{V}_{\mathrm{r}}),\ \nu, \nu' \in \mathit{Val},\ (x, \nu)\mathscr{R}(x', \nu'),\ (y, \nu)\mathscr{S}(y', \nu')\big\} \cup \mathscr{U}_{\parallel},$$

$$\mathscr{U}_{|} = \big\{((x \mid y, \nu), (x' \mid y', \nu')) \mid x, x', y, y' \in \mathcal{T}(\mathcal{V}_{\mathrm{r}}),\ \nu, \nu' \in \mathit{Val},\ (x, \nu)\mathscr{R}(x', \nu'),\ (y, \nu)\mathscr{S}(y', \nu')\big\} \cup \mathscr{U}_{\parallel},$$

$$\mathscr{U}_{d\gg} = \big\{((d \gg x, \nu), (d \gg x', \nu')) \mid x, x' \in \mathcal{T}(\mathcal{V}_{\mathrm{r}}),\ \nu, \nu' \in \mathit{Val},\ (x, \nu)\mathscr{R}(x', \nu')\big\} \cup \mathscr{R},$$

$$\mathscr{U}_{\partial_H()} = \big\{((\partial_H(x), \nu), (\partial_H(x'), \nu')) \mid x, x' \in \mathcal{T}(\mathcal{V}_{\mathrm{r}}),\ \nu, \nu' \in \mathit{Val},\ (x, \nu)\mathscr{R}(x', \nu')\big\}.$$

□

Appendix B. Stateless bisimilarity

In the proof of Theorem 9, we claimed that the notion of robust bisimilarity coincides with the notion of bisimilarity used in [46]. In this appendix, we substantiate that claim. Firstly, the notion of bisimilarity on process terms from [46], is defined as follows:

Definition B.1 (*Stateless bisimilarity*). A relation $\mathscr{R} \subseteq \mathcal{T}(\mathcal{V}_{\mathrm{r}}) \times \mathcal{T}(\mathcal{V}_{\mathrm{r}})$ on process terms, is a *stateless bisimulation* relation if for all $p, q \in \mathcal{T}(\mathcal{V}_{\mathrm{r}})$ such that $p\mathscr{R}q$, and for all valuations $\nu, \nu' \in \mathit{Val}$ and labels $l \in A \cup \Sigma$, we find

- $\langle p, \nu \rangle \checkmark$ implies $\langle q, \nu \rangle \checkmark$;
- $\langle q, \nu \rangle \checkmark$ implies $\langle p, \nu \rangle \checkmark$;
- for every p' with $\langle p, \nu \rangle \xrightarrow{l} \langle p', \nu' \rangle$ there exists q' s.t. $\langle q, \nu \rangle \xrightarrow{l} \langle q', \nu' \rangle$ and $p'\mathscr{R}q'$;
- for every q' with $\langle q, \nu \rangle \xrightarrow{l} \langle q', \nu' \rangle$ there exists p' s.t. $\langle p, \nu \rangle \xrightarrow{l} \langle p', \nu' \rangle$ and $p'\mathscr{R}q'$.

Two process terms x and y are *stateless bisimilar*, denoted $x \underline{\leftrightarrow}_s y$, if there exists a stateless bisimulation relation that relates them.

Now we will show that the two notions coincide.

Theorem B.2. *For all process terms $p, q \in \mathcal{T}(\mathcal{V}_{\mathrm{r}})$, we find $p \underline{\leftrightarrow} q$ iff $p \underline{\leftrightarrow}_s q$.*

Proof. We start by showing that $\underline{\leftrightarrow}_s \subseteq \underline{\leftrightarrow}$. In order to do this, suppose that two process terms p and q are stateless bisimilar ($p \underline{\leftrightarrow}_s q$), and that $\mathscr{R}$ is a relation that witnesses this equivalence. Then we define a relation $\mathscr{S} = \{((x, \nu), (y, \nu)) \mid x\mathscr{R}y,\ \nu \in \mathit{Val}\}$. It is straightforward to verify that this a bisimulation relation in the sense of this paper, and furthermore, if $(x, \nu)\mathscr{S}(y, \nu')$, then $\nu = \nu'$ and hence $\iota(\nu) = \iota(\nu')$ for every interference ι. Finally, we observe that $(x, \nu'')\mathscr{S}(y, \nu'')$ for every $\nu'' \in \mathit{Val}$, and particularly for every $\iota(\nu)$. Hence $\mathscr{S}$ is robust, and witnesses $p \underline{\leftrightarrow} q$.

Now, we will show that $\underline{\leftrightarrow} \subseteq \underline{\leftrightarrow}_s$. Suppose that we have process terms p and q that are robustly bisimilar ($p \underline{\leftrightarrow} q$), and that $\mathscr{S}$ is a robust bisimulation relation that witnesses this. Then, we construct the relation $\mathscr{R} = \{(x, y) \mid \forall_\nu (x, \nu)\mathscr{S}(y, \nu)\}$. Clearly, $p\mathscr{R}q$, since we have $(p, \nu)\mathscr{S}(q, \nu)$ for every ν. The case for termination, is also straightforward. Finally, suppose that $x\mathscr{R}y$, and there exists a transition $\langle x, \nu \rangle \xrightarrow{l} \langle x', \nu' \rangle$. Then, by definition of $\mathscr{R}$ we know $(x, \nu)\mathscr{S}(y, \nu)$, and because $\mathscr{S}$ is a bisimulation relation we find that there is a

transition $\langle y, \nu \rangle \xrightarrow{l} \langle y', \nu'' \rangle$. Using Lemma A.1, from Appendix A, we find that $\nu' = \nu''$, and hence that $(x', \nu')\mathscr{S}(y', \nu')$. Using this, we can construct for every μ an interference ι such that $\iota(\nu') = \mu$, and using robustness we conclude that $(x', \mu)\mathscr{S}(y', \mu)$. From this it follows that $x'\mathscr{R}y'$, which proves that $\mathscr{R}$ is a stateless bisimulation relation, witnessing $p \leftrightarrows_s q$. □

Appendix C. Soundness

In this section, we summarize the proofs for soundness of the axiomatization and of the derivation rules, as given in [46]. The complete proofs are very long, but rather straightforward, and are given in [46] for a notion of stateless bisimilarity, that has been proven to coincide with robust bisimilarity in Appendix B. In this section, we will confine ourselves to give only the witnessing robust bisimulation relations for some of the more difficult derivation rules and axioms. Two of the axioms are worked out in more detail.

Soundness of derivation rules (1)–(3) follows directly from the fact that robust bisimilarity is an equivalence relation. That bisimilarity is an equivalence is a standard result [55], and that robustness does not change this, is easy to verify.

Derivation rules (4) and (5) are sound, because robust bisimilarity is a congruence for all the operators of HyPA. This is proven in Appendix A.

Soundness of derivation rules (6) and (7) is straightforward from the operational semantics of re-initialization clauses and flow-clauses, while soundness of derivation rule (8) follows from soundness of all the axioms separately, and from the fact that the semantics of a recursive definition indeed reflect a solution of the recursive equation.

Soundness of derivation rule (9), is witnessed by a relation $\mathscr{R}$ such that $(d \gg c, \nu)\ \mathscr{R}\ (d \gg c' \rhd c, \nu) \wedge (c, \nu)\ \mathscr{R}\ (c' \blacktriangleright c, \nu) \wedge (x, \nu)\ \mathscr{R}\ (x, \nu)$, for all $\nu \in \mathit{Val}$, $x \in \mathcal{T}(\mathcal{V}_r)$ and all $c, c' \in C$ that satisfy the assumption that $(\mu, \sigma) \models c'$ implies $(\mu, \sigma) \models c$ and, $(\mu, \mu') \models d$ and $(\mu', \sigma) \models c$ implies $(\mu', \sigma) \models c'$. To verify that this is indeed a robust bisimulation relation, is straightforward.

Soundness of derivation rule (10) is witnessed by the relation $\mathscr{R}$ such that $(c, \nu)\ \mathscr{R}\ ((c' \oplus c'') \rhd c, \nu) \wedge (c, \nu)\ \mathscr{R}\ (c' \blacktriangleright c, \nu) \wedge (c, \nu)\ \mathscr{R}\ (c'' \blacktriangleright c, \nu) \wedge (x, \nu)\ \mathscr{R}\ (x, \nu)$ for all $\nu \in \mathit{Val}$, $x \in \mathcal{T}(\mathcal{V}_r)$ and all $c, c', c'' \in C$ satisfying the assumption that $(\mu, \sigma) \models c$ if and only if $(\mu, \sigma) \models c'$ or $(\mu, \sigma) \models c''$. Again, it is straightforward to verify that this is a robust bisimulation relation.

As examples of soundness proofs of the axioms, we have selected a few axioms that we study in more detail. The witnessing relations for all the others, and the proofs that these relations are indeed bisimulation relations, can be found in [46] for the notion of stateless bisimilarity. The translation to robust bisimilarity is straightforward using the results of Appendix B.

The first axiom we give a witness relation for, regards distribution of disrupt over sequential composition. It is the only axiom that was not mentioned in [46]. The axiom $(x \odot \delta \rhd y) \odot z \approx x \odot \delta \rhd y \odot z$ is witnessed by the relation $\mathscr{R}$ such that $((x \odot \delta \rhd y) \odot z, \nu)\ \mathscr{R}\ (x \odot \delta \rhd y \odot z, \nu)$, $((x \odot \delta \blacktriangleright y) \odot z, \nu)\ \mathscr{R}\ (x \odot \delta \blacktriangleright y \odot z, \nu)$ and $(x, \nu)\ \mathscr{R}\ (x, \nu)$ for all $x, y, z \in \mathcal{T}(\mathcal{V}_r)$ and $\nu \in \mathit{Val}$. That this is indeed a robust bisimulation relation is straightforward to verify.

The axiom $d \gg \epsilon \mid d' \gg \epsilon \approx (d^? \wedge d'^?) \gg \epsilon$ is witnessed by the relation $\mathscr{R}$ such that $(d \gg \epsilon \mid d' \gg \epsilon, \nu)\, \mathscr{R}\, ((d^? \wedge d'^?) \gg \epsilon, \nu)$, for all $\nu \in \mathit{Val}$ and $d, d' \in D$. Since this is one of the more difficult axioms, we show the full proof here. Clearly, we only need to verify bisimilarity for the cases of $(d \gg \epsilon \mid d' \gg \epsilon, \nu)\, \mathscr{R}\, ((d^? \wedge d'^?) \gg \epsilon, \nu)$ for termination. Furthermore, it is obvious from the construction of $\mathscr{R}$ that it is a robust relation.

(1) $\langle d \gg \epsilon \mid d' \gg \epsilon, \nu \rangle \checkmark$, for which we need the hypothesis
   (a) $\exists_{\nu'} (\nu, \nu') \models d \wedge \exists_{\nu''} (\nu, \nu'') \models d'$
   From which we conclude that $(\nu, \nu) \models d^?$ and $(\nu, \nu) \models d'^?$, hence $\langle (d^? \wedge d'^?) \gg \epsilon, \nu \rangle \checkmark$.

(2) $\langle (d^? \wedge d'^?) \gg \epsilon, \nu \rangle \checkmark$, for which we need the hypothesis
   (a) $\exists_{\nu'} (\nu, \nu') \models (d^? \wedge d'^?)$, which comes down to the hypothesis
      (i) $\nu = \nu' \wedge \exists_{\upsilon} (\nu, \upsilon) \models d \wedge \exists_{\upsilon'} (\nu, \upsilon') \models d'$
      From which we easily conclude $\langle d \gg \epsilon, \nu \rangle \checkmark$ and $\langle d' \gg \epsilon, \nu \rangle \checkmark$, hence $\langle d \gg \epsilon \mid d' \gg \epsilon, \nu \rangle \checkmark$.

The axiom $d \gg c \rhd x \mid d' \gg c' \rhd y \approx ((d \sim c_{jmp}) \wedge (d' \sim c'_{jmp})) \gg (c \wedge c') \rhd \big(x \between c' \blacktriangleright y \oplus y \between c \blacktriangleright x \oplus x \mid c' \blacktriangleright y \oplus y \mid c \blacktriangleright x\big)$ is witnessed by the relation $\mathscr{R}$ such that $(d \gg c \rhd x \mid d' \gg c' \rhd y, \nu)\, \mathscr{R}\, (N \gg c \wedge c' \rhd M, \nu) \wedge (c \blacktriangleright x \parallel c' \blacktriangleright y, \nu)\, \mathscr{R}\, (c \wedge c' \blacktriangleright M, \nu) \wedge (x \parallel y, \nu)\, \mathscr{R}\, (y \parallel x, \nu) \wedge (x, \nu)\, \mathscr{R}\, (x, \nu)$, for all $\nu \in \mathit{Val}$, $c, c' \in C$, $d, d' \in D$ and $x, y \in \mathcal{T}(\mathcal{V}_\mathrm{r})$, in which we use abbreviations $M = x \between c' \blacktriangleright y \oplus y \between c \blacktriangleright x \oplus x \mid c' \blacktriangleright y \oplus y \mid c \blacktriangleright x$ and $N = ((d \sim c_{jmp}) \wedge (d' \sim c'_{jmp}))$. The proof that this is a bisimulation relation, is rather complicated, and therefore we give it below. That it is a robust relation, follows straightforwardly from the construction.

In the proof below, we make use of the following two lemmas, which are proven in [46]. These lemmas express that the initial jumps that a flow-clause can make, are closed under concatenation, and that it is not necessary (yet still possible) to jump if there is a solution that starts from the current valuation. This is vital, since the axiom expresses that any number of re-initializations c_{jmp} may be performed before actually executing a flow transition. Incidentally, these lemmas are also needed for the proof of the axiom $c_{jmp} \gg c \approx c$, in which they are used in a similar way as in the proof below.

Lemma C.1. *If* $(\nu, \sigma') \models c$ *and* $(\sigma'(0), \sigma) \models c$ *then* $(\nu, \sigma) \models c$.

Lemma C.2. *If* $(\nu, \sigma) \models c$ *then* $(\sigma(0), \sigma) \models c$.

The validity of these lemmas does not depend on the choice of parameters of HyPA, but follows directly from the operational semantics.

For $(x, \nu)\, \mathscr{R}\, (x, \nu)$, the proof that $\mathscr{R}$ is a bisimulation relation is trivial.

For $(x \parallel y, \nu)\, \mathscr{R}\, (y \parallel x, \nu)$, the proof is also straightforward.

For $(d \gg c \rhd x \mid d' \gg c' \rhd y, \nu)\, \mathscr{R}\, (N \gg c \wedge c' \rhd M, \nu)$, we find the following cases:

(1) $\langle d \gg c \rhd x \mid d' \gg c' \rhd y, \nu \rangle \checkmark$, for which we need the hypothesis
   (a) $\langle d \gg c \rhd x, \nu \rangle \checkmark \wedge \langle d' \gg c' \rhd y, \nu \rangle \checkmark$, which leads to the hypothesis
      (i) $\exists_{\nu'} (\nu, \nu') \models d \wedge \langle c \rhd x, \nu' \rangle \checkmark$, for which we need the hypothesis
         (A) $\langle c, \nu' \rangle \checkmark$, which cannot be satisfied.

(2) $\langle N \gg c \wedge c' \rhd M, \nu \rangle \checkmark$, cannot be satisfied for similar reasons as in the previous case.

(3) $\langle d \gg c \rhd x \mid d' \gg c' \rhd y, \nu \rangle \xrightarrow{l} \langle p, \nu''' \rangle$, leading to one of the hypotheses

(a) $\exists_{p'}\ l \in A \ \wedge\ \langle d \gg c \rhd x, \nu \rangle \overset{l}{\mapsto} \langle p', \nu''' \rangle$, which can clearly not be satisfied since flow-clauses cannot execute action transitions.

(b) $\exists_{p',p''}\ l \in \Sigma \ \wedge\ dom(l) = [[0,t] \ \wedge\ p = p' \parallel p'' \ \wedge\ \langle d \gg c \rhd x, \nu \rangle \overset{l}{\rightsquigarrow} \langle p', \nu''' \rangle \ \wedge\ \langle d' \gg c' \rhd y, \nu \rangle \overset{l}{\rightsquigarrow} \langle p'', \nu''' \rangle$, for which we need the hypothesis

(i) $\exists_{\nu'}(\nu, \nu') \models d \ \wedge\ \langle c \rhd x, \nu' \rangle \overset{l}{\rightsquigarrow} \langle p', \nu''' \rangle \ \wedge\ \exists_{\nu''}\ (\nu, \nu'') \models d' \ \wedge\ \langle c' \rhd y, \nu'' \rangle \overset{l}{\rightsquigarrow} \langle p'', \nu''' \rangle$, leading to the hypothesis

(A) $\exists_{r'}\ p' = r' \blacktriangleright x \ \wedge\ \langle c, \nu' \rangle \overset{l}{\rightsquigarrow} \langle r', \nu''' \rangle \ \wedge\ \exists_{r''}\ p'' = r'' \blacktriangleright y \ \wedge\ \langle c', \nu'' \rangle \overset{l}{\rightsquigarrow} \langle r'', \nu''' \rangle$, for which we need the hypothesis

- $(\nu', l) \models c \ \wedge\ r' = c \ \wedge\ (\nu'', l) \models c' \ \wedge\ r'' = c' \ \wedge\ \nu''' = l(t)$.
  Using Lemma C.2 we find that $(l(0), l) \models (c \wedge c')$. Furthermore, we may conclude that $(\nu, l(0)) \models N$ and $p = c \blacktriangleright x \parallel c' \blacktriangleright y$, to finally find $\langle N \gg c \wedge c' \rhd M, \nu \rangle \xrightarrow{l} \langle (c \wedge c') \blacktriangleright M, \nu''' \rangle$ and $(p, \nu''') \mathscr{R} (c \wedge c' \blacktriangleright M, \nu''')$.

(4) $\langle N \gg (c \wedge c') \rhd M, \nu \rangle \xrightarrow{l} \langle p, \nu'' \rangle$, leading to the hypothesis

(a) $\exists_{\nu'}\ (\nu, \nu') \models N \ \wedge\ \langle (c \wedge c') \rhd M, \nu' \rangle \xrightarrow{l} \langle p, \nu'' \rangle$, for which we need the hypothesis

(i) $\exists_r p = r \blacktriangleright M \ \wedge\ \langle (c \wedge c'), \nu' \rangle \xrightarrow{l} \langle r, \nu'' \rangle \ \wedge\ \exists_{\nu_1,\sigma_1}(\nu, \nu_1) \models d \ \wedge\ (\nu_1, \sigma_1) \models c \ \wedge\ \exists_{\nu_2,\sigma_2}\ (\nu, \nu_2) \models d' \ \wedge\ (\nu_2, \sigma_2) \models c' \ \wedge\ \nu' = \sigma_1(0) = \sigma_2(0)$, and finally we need the hypothesis

(A) $l \in \Sigma \ \wedge\ r = (c \wedge c') \ \wedge\ (\nu', l) \models (c \wedge c') \ \wedge\ \nu'' = l(t)$.
From this we may conclude that $p = (c \wedge c') \blacktriangleright M$, but furthermore we can use Lemma C.1, together with the facts that $(\nu_1, \sigma_1) \models c$ and $(\nu', l) \models c$ and $\nu' = \sigma_1(0)$ to find $(\nu_1, l) \models c$ and similarly $(\nu_2, l) \models c'$. This leads to the observations that $\langle d \gg c \rhd x \rangle \overset{l}{\rightsquigarrow} \langle c \blacktriangleright x, \nu'' \rangle$ and $\langle d' \gg \rhd c' y \rangle \overset{l}{\rightsquigarrow} \langle c' \blacktriangleright y, \nu'' \rangle$, and finally $\langle d \gg c \rhd x \mid d' \gg c' \rhd y, \nu \rangle \xrightarrow{l} \langle c \blacktriangleright x \parallel c \blacktriangleright y, \nu'' \rangle$ and $(c \blacktriangleright x \parallel c \blacktriangleright y, \nu'') \mathscr{R} (p, \nu'')$.

For $(c \blacktriangleright x \parallel c' \blacktriangleright y, \nu) \mathscr{R} (c \wedge c' \blacktriangleright M, \nu)$, we find the following cases.

(1) $\langle c \blacktriangleright x \parallel c' \blacktriangleright y, \nu \rangle \checkmark$, for which we need the hypothesis

(a) $\langle c \blacktriangleright x, \nu \rangle \checkmark \ \wedge\ \langle c' \blacktriangleright y, \nu \rangle \checkmark$, for which we need the hypothesis

(i) $\langle x, \nu \rangle \checkmark \ \wedge\ \langle y, \nu \rangle \checkmark$
From which we may conclude $\langle x \mid y, \nu \rangle \checkmark$ hence $\langle M, \nu \rangle \checkmark$ and $\langle (c \wedge c') \blacktriangleright M, \nu \rangle \checkmark$.

(2) $\langle c \wedge c' \blacktriangleright M, \nu \rangle \checkmark$, for which we need the hypothesis $\langle M, \nu \rangle \checkmark$ and hence one of the following hypotheses

(a) $\langle x \between c' \blacktriangleright y, \nu \rangle \checkmark$, which cannot occur.

(b) $\langle y \between c \blacktriangleright x, \nu \rangle \checkmark$, which cannot occur.

(c) $\langle x \mid c' \blacktriangleright y, \nu \rangle \checkmark$, for which we need the hypothesis

(i) $\langle x \rangle \checkmark \ \wedge\ \langle c' \blacktriangleright y, \nu \rangle \checkmark$. From this we may conclude that $\langle c \blacktriangleright x, \nu \rangle \checkmark$ and hence $\langle c \blacktriangleright x \parallel c' \blacktriangleright y, \nu \rangle \checkmark$.

(d) $\langle y \mid c \blacktriangleright x, \nu \rangle \checkmark$, is similar to the previous case.

(3) $\langle c \blacktriangleright x \parallel c' \blacktriangleright y, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$, for which we need one of the following hypotheses:

(a) $\exists_{a,a',p',p''}\ l = (a\gamma a', \mu) \ \wedge\ p = p' \parallel p'' \ \wedge\ \langle c \blacktriangleright x, \nu \rangle \overset{a,\mu}{\mapsto} \langle p', \nu' \rangle \ \wedge\ \langle c' \blacktriangleright y, \nu \rangle \overset{a',\mu}{\mapsto} \langle p'', \nu' \rangle$, which leads to the hypothesis

(i) $\langle x, \nu \rangle \overset{a,\mu}{\mapsto} \langle p', \nu' \rangle \ \wedge\ \langle y, \nu \rangle \overset{a',\mu}{\mapsto} \langle p'', \nu' \rangle$,

from which we conclude $\langle x \mid c' \blacktriangleright y, \nu \rangle \overset{a\gamma a',\mu}{\mapsto} \langle p' \parallel p'', \nu' \rangle$, and hence $\langle (c \wedge c') \blacktriangleright M, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$ with $(p, \nu')\, \mathscr{R}\, (p, \nu')$.

(b) $\exists_{p'}\ l \in A \ \wedge\ p = p' \parallel c' \blacktriangleright y \wedge \langle c \blacktriangleright x \rangle \overset{l}{\mapsto} \langle p', \nu' \rangle$, for which we need the hypothesis

(i) $\langle x, \nu \rangle \overset{l}{\mapsto} \langle p', \nu' \rangle$

from which we conclude that $\langle x \between c' \blacktriangleright y, \nu \rangle \xrightarrow{l} \langle p' \parallel c' \blacktriangleright y, \nu' \rangle$ and hence $\langle (c \wedge c') \blacktriangleright M, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$ with $(p, \nu')\, \mathscr{R}\, (p, \nu')$.

(c) $\exists_{p'}\ l \in A \ \wedge\ p = c \blacktriangleright x \parallel p' \wedge \langle c' \blacktriangleright y \rangle \overset{l}{\mapsto} \langle p', \nu' \rangle$, which is similar to the previous case.

(d) $\exists_{p',p'',t}\ l \in \Sigma \ \wedge\ dom(l) = [[0, t] \ \wedge\ p = p' \parallel p'' \ \wedge\ \langle c \blacktriangleright x, \nu \rangle \overset{l}{\rightsquigarrow} \langle p', \nu' \rangle \ \wedge\ \langle c' \blacktriangleright y, \nu \rangle \overset{l}{\rightsquigarrow} \langle p'', \nu' \rangle$, for which we need one of the following hypotheses:

(i) $\exists_{r'}\ p' = r' \blacktriangleright x \ \wedge\ \langle c, \nu \rangle \overset{l}{\rightsquigarrow} \langle r', \nu' \rangle \ \wedge\ \exists_{r''}\ p'' = r'' \blacktriangleright y \ \wedge\ \langle c', \nu \rangle \overset{l}{\rightsquigarrow} \langle r'', \nu' \rangle$, for which we need the hypothesis

(A) $r' = c \ \wedge\ r'' = c' \ \wedge\ (\nu, l) \models c \ \wedge\ (\nu, l) \models c' \ \wedge\ \nu' = l(t)$

From this we conclude that $p = c \blacktriangleright x \parallel c' \blacktriangleright y$ and $\langle (c \wedge c') \blacktriangleright M, \nu \rangle \xrightarrow{l} \langle c \wedge c' \blacktriangleright M, \nu' \rangle$, with $(p, \nu')\, \mathscr{R}\, ((c \wedge c') \blacktriangleright M, \nu')$.

(ii) $\exists_{r'}\ p' = r' \blacktriangleright x \ \wedge\ \langle c, \nu \rangle \overset{l}{\rightsquigarrow} \langle r', \nu' \rangle \ \wedge\ \langle y, \nu \rangle \overset{l}{\rightsquigarrow} \langle p'', \nu' \rangle$, for which we need the hypothesis

(A) $r' = c$.

Now, we conclude that $p = c \blacktriangleright x \parallel p''$, and that $\langle y \mid c \blacktriangleright x, \nu \rangle \xrightarrow{l} \langle p'' \parallel c \blacktriangleright x, \nu' \rangle$. Hence $\langle (c \wedge c') \blacktriangleright M, \nu \rangle \xrightarrow{l} \langle p'' \parallel c \blacktriangleright x, \nu' \rangle$ with $(p'' \parallel c \blacktriangleright x, \nu')\, \mathscr{R}\, (p, \nu')$.

(iii) $\langle x, \nu \rangle \overset{l}{\rightsquigarrow} \langle p', \nu' \rangle \ \wedge\ \exists_{r''}\ p'' = r'' \blacktriangleright y \ \wedge\ \langle c', \nu \rangle \overset{l}{\rightsquigarrow} \langle r'', \nu' \rangle$, for which we need the hypothesis

(A) $r'' = c'$.

Now, we conclude that $p = p' \parallel c' \blacktriangleright y$, and that $\langle x \mid c' \blacktriangleright y, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$. Hence, $\langle (c \wedge c') \blacktriangleright M, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$ with $p\, \mathscr{R}\, p$.

(iv) $\langle x, \nu \rangle \overset{l}{\rightsquigarrow} \langle p', \nu' \rangle \ \wedge\ \langle y, \nu \rangle \overset{l}{\rightsquigarrow} \langle p'', \nu' \rangle$

From which it follows directly that $\langle x \mid c' \blacktriangleright y \rangle \xrightarrow{l} \langle p, \nu' \rangle$ Hence, $\langle (c \wedge c') \blacktriangleright M, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$ with $(p, \nu')\, \mathscr{R}\, (p, \nu')$.

(e) $l \in \Sigma \ \wedge\ \langle c \blacktriangleright x, \nu \rangle \overset{l}{\rightsquigarrow} \langle p, \nu' \rangle \ \wedge\ \langle c' \blacktriangleright y, \nu \rangle \checkmark$, for which we need the hypothesis

(i) $\langle y, \nu \rangle \checkmark$.

From this we may conclude $\langle y \mid c \blacktriangleright x \rangle \xrightarrow{l} \langle p, \nu' \rangle$. Hence, $\langle (c \wedge c') \blacktriangleright M, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$ with $(p, \nu') \mathrel{\mathscr{R}} (p, \nu')$.

(f) $l \in \Sigma \wedge \langle c \blacktriangleright x, \nu \rangle \checkmark \wedge \langle c' \blacktriangleright y, \nu \rangle \overset{l}{\rightsquigarrow} \langle p, \nu' \rangle$, for which we need the hypothesis

(i) $\langle x, \nu \rangle \checkmark$.

From this we may conclude $\langle x \mid c' \blacktriangleright y \rangle \xrightarrow{l} \langle p, \nu' \rangle$. Hence, $\langle (c \wedge c') \blacktriangleright M, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$ with $(p, \nu') \mathrel{\mathscr{R}} (p, \nu')$.

(4) $\langle c \wedge c' \blacktriangleright M, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$, which needs one of the following hypotheses:

(a) $\exists_r\ p = r \blacktriangleright M \wedge \langle (c \wedge c'), \nu \rangle \xrightarrow{l} \langle r, \nu' \rangle$, for which we need the hypothesis

(i) $\exists_t\ l \in \Sigma \wedge dom(l) = [[0, t] \wedge r = (c \wedge c') \wedge \nu' = l(t) \wedge (\nu, l) \models c \wedge (\nu, l) \models c'$.

From this we may readily conclude that $p = (c \wedge c') \blacktriangleright M$ and $\langle c \blacktriangleright x, \nu \rangle \overset{\sigma}{\rightsquigarrow} \langle c \blacktriangleright x, \nu' \rangle$. Consequently, we find $\langle c \blacktriangleright x \parallel c' \blacktriangleright y, \nu \rangle \xrightarrow{l} \langle c \blacktriangleright x \parallel c' \blacktriangleright y, \nu' \rangle$ with $(c \blacktriangleright x \parallel c' \blacktriangleright y, \nu') \mathrel{\mathscr{R}} (p, \nu')$.

(b) $\langle M, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$, which comes down to one of the hypotheses:

(i) $\langle x \between c' \blacktriangleright y, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$, for this we need the hypothesis

(A) $\exists_r\ l \in A \wedge p = r \parallel c' \blacktriangleright y \wedge \langle x, \nu \rangle \overset{l}{\mapsto} \langle r, \nu' \rangle$.

From which we conclude $\langle c \blacktriangleright x \rangle \overset{l}{\mapsto} \langle r, \nu' \rangle$ and finally $\langle c \blacktriangleright x \parallel c' \blacktriangleright y, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$ with $(p, \nu') \mathrel{\mathscr{R}} (p, \nu')$.

(ii) $\langle y \between c \blacktriangleright x, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$, for this we need the hypothesis

(A) $\exists_r\ l \in A \wedge p = r \parallel c \blacktriangleright x \wedge \langle y, \nu \rangle \overset{l}{\mapsto} \langle r, \nu' \rangle$.

From which we conclude $\langle c \blacktriangleright y \rangle \overset{l}{\mapsto} \langle r, \nu' \rangle$ and finally $\langle c \blacktriangleright x \parallel c' \blacktriangleright y, \nu \rangle \xrightarrow{l} \langle c \blacktriangleright x \parallel r, \nu' \rangle$ with $(p, \nu') \mathrel{\mathscr{R}} (c \blacktriangleright x \parallel r, \nu')$.

(iii) $\langle x \mid c' \blacktriangleright y, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$, for which we need one of the hypotheses

(A) $\exists_{a,a',p',p'',\mu}\ l = (a \gamma a', \mu) \wedge p = p' \parallel p'' \wedge \langle x, \nu \rangle \overset{a,\mu}{\mapsto} \langle p', \nu' \rangle \wedge \langle c' \blacktriangleright y, \nu \rangle \overset{a',\mu}{\mapsto} \langle p'', \nu' \rangle$, which leads to the hypothesis

- $\langle y, \nu \rangle \overset{a',\mu}{\mapsto} \langle p'', \nu' \rangle$

From which we readily conclude $\langle c \blacktriangleright x \parallel c' \blacktriangleright y, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$ with $(p, \nu') \mathrel{\mathscr{R}} (p, \nu')$.

(B) $\exists_{p',p''}\ l \in \Sigma \wedge p = p' \parallel p'' \wedge \langle x, \nu \rangle \overset{l}{\rightsquigarrow} \langle p', \nu' \rangle \wedge \langle c' \blacktriangleright y, \nu \rangle \overset{l}{\rightsquigarrow} \langle p'', \nu' \rangle$, which leads to one of the hypotheses

- $\exists_r\ p'' = r \blacktriangleright y \wedge \langle c' \blacktriangleright y, \nu \rangle \overset{l}{\rightsquigarrow} \langle r, \nu' \rangle$, then we need the hypothesis
  - $r = c'$

From which we conclude that $p = p' \parallel c' \blacktriangleright y$ and $\langle c \blacktriangleright x \parallel c' \blacktriangleright y, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$ with $(p, \nu') \mathrel{\mathscr{R}} (p, \nu')$.

- $\langle y, \nu \rangle \overset{l}{\rightsquigarrow} \langle p'', \nu' \rangle$

From which we readily conclude $\langle c \blacktriangleright x \parallel c' \blacktriangleright y, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$ with $(p, \nu') \mathrel{\mathscr{R}} (p, \nu')$.

(iv) $\langle y \mid c \blacktriangleright x, \nu \rangle \xrightarrow{l} \langle p, \nu' \rangle$, for which we need one of the hypotheses

(A) $\exists_{a,a',p',p'',\mu}\ l = (a\gamma a', \mu)\ \ p = p' \parallel p''\ \ \wedge\ \ \langle y, \nu \rangle \overset{a,\mu}{\mapsto} \langle p', \nu' \rangle\ \ \wedge\ \ \langle c \blacktriangleright x, \nu \rangle \overset{a',\mu}{\mapsto} \langle p'', \nu' \rangle$, which leads to the hypothesis

- $\langle x, \nu \rangle \overset{a',\mu}{\mapsto} \langle p'', \nu' \rangle$

  From which we readily conclude $\langle c \blacktriangleright x \parallel c' \blacktriangleright y, \nu \rangle \xrightarrow{l} \langle p'' \parallel p', \nu' \rangle$ with $(p, \nu') \mathrel{\mathscr{R}} (p'' \parallel p', \nu')$.

(B) $\exists_{p',p''}\ l \in \Sigma\ \wedge\ p = p' \parallel p''\ \wedge\ \langle y, \nu \rangle \overset{l}{\rightsquigarrow} \langle p', \nu' \rangle\ \wedge\ \langle c' \blacktriangleright x, \nu \rangle \overset{l}{\rightsquigarrow} \langle p'', \nu' \rangle$, which leads to one of the hypotheses

- $\exists_r\ p'' = r \blacktriangleright x\ \wedge\ \langle c \blacktriangleright x, \nu \rangle \overset{l}{\rightsquigarrow} \langle r, \nu' \rangle$, then we need the hypothesis
  - $r = c$

  From which we conclude that $p = p' \parallel c \blacktriangleright x$ and $\langle c \blacktriangleright x \parallel c' \blacktriangleright y, \nu \rangle \xrightarrow{l} \langle c \blacktriangleright x \parallel p', \nu' \rangle$ with $(p, \nu') \mathrel{\mathscr{R}} (c \blacktriangleright x \parallel p', \nu')$.
- $\langle x, \nu \rangle \overset{l}{\rightsquigarrow} \langle p'', \nu' \rangle$

  From which we readily conclude $\langle c \blacktriangleright x \parallel c' \blacktriangleright y, \nu \rangle \xrightarrow{l} \langle p'' \parallel p', \nu' \rangle$ with $(p, \nu') \mathrel{\mathscr{R}} (p'' \parallel p', \nu')$.

Appendix D. Recursion principles

The *recursive specification principle* RSP states that a guarded recursive specification has at most one solution. Formally, the rule is stated as follows:

$$\frac{S \models E,\ S' \models E,\ E \text{ guarded}}{S(X) \approx S'(X)}\ X \in \mathcal{V}_{\mathrm{r}},$$

where, $S \models E$ denotes that the interpretation $S \in \mathcal{V}_{\mathrm{r}} \to \mathcal{T}(\mathcal{V}_{\mathrm{r}})$ of recursion variables is a solution of a guarded recursive specification E. The proof of this, usually goes via another principle, called the *approximation induction principle* AIP [49], which makes use of a family of projection operators π_n. AIP states that if every finite projection of two processes is bisimilar, then the two processes are bisimilar. For the kind of semantical model we use, AIP is restricted in the sense that one of the compared processes should have bounded non-determinism. This is usually referred to as the *restricted approximation induction principle* AIP$^-$. In this section, we introduce the family of projection operators, and formalize the notion of bounded non-determinism. Then we pose the approximation induction principle, and prove it sound. After that, we show the existence of a bounded solution for guarded recursive specifications, and prove a projection property for guarded process terms. Finally, this allows us to prove soundness of RSP using AIP$^-$.

Projection has the following operational semantics:

$$\frac{\langle p, \nu \rangle \surd}{\langle \pi_n(p), \nu \rangle \surd}, \qquad \frac{\langle p, \nu \rangle \xrightarrow{l} \langle p', \nu' \rangle}{\langle \pi_{n+1}(p), \nu \rangle \xrightarrow{l} \langle \pi_n(p'), \nu' \rangle}.$$

Without proof, we claim that robust bisimilarity is a congruence for projection. Bounded non-determinism $B(p)$ is defined as follows.

Definition D.1 (*Bounded non-determinism*). Bounded non-determinism is recursively defined as:

- Every state has bounded non-determinism in 0 steps.
- A state (p, ν) has bounded non-determinism in $n+1$ steps, if for every l the set $R = \{(p', \nu') \mid \langle\, p, \nu \,\rangle \xrightarrow{l} \langle\, p', \nu' \,\rangle\}$ is finite, and all elements $(p', \nu') \in R$ have bounded non-determinism in n steps themselves.
- A state (p, ν) has bounded non-determinism (denoted $B(p, \nu)$) if it has bounded non-determinism for any arbitrary number of steps.
- A process p has bounded non-determinism (denoted $B(p)$) if for every valuation $\nu \in \mathit{Val}$ we find that (p, ν) has bounded non-determinism.

These definitions allow us to state the restricted approximation induction principle AIP$^-$:

$$\frac{\forall_n \pi_n(p) \approx \pi_n(q) \;\wedge\; B(q)}{p \approx q}\ \text{AIP}^-$$

Next, we prove that this principle is sound.

Theorem D.2. *AIP$^-$ is sound for the semantics of HyPA.*

Proof. To prove this principle sound, suppose that $\mathscr{R}$ is the union of all robust bisimulation relations. In particular, it contains the robust bisimulation relations witnessing $\pi_n(p) \leftrightarrows \pi_n(q)$. Note, that $\mathscr{R}$ is an equivalence relation on states.

We now construct the following relation:

$$\mathscr{S} = \big\{((x, \nu), (y, \mu)) \mid \forall_n\ (\pi_n(x), \nu)\mathscr{R}(\pi_n(y), \mu),\ B(y, \nu)\big\},$$

and show that this is a robust bisimulation relation witnessing $p \leftrightarrows q$.

It is obvious that for all ν and all n we have $(\pi_n(p), \nu)\mathscr{R}(\pi_n(q), \nu)$, therefore we know $(p, \nu)\mathscr{S}(q, \nu)$. So, if $\mathscr{S}$ is a robust bisimulation relation, then it is a witness.

In order to verify that $\mathscr{S}$ is a bisimulation relation, assume $(x, \nu)\mathscr{S}(y, \mu)$ and study the following cases:

(1) $\langle\, x, \nu \,\rangle\checkmark$. Using the semantics of projection, we find $\langle\, \pi_n(x), \nu \,\rangle\checkmark$ for all n, and using the definition of $\mathscr{S}$ we get $(\pi_n(x), \nu)\mathscr{R}(\pi_n(y), \mu)$.
From which we conclude, using the fact that $\mathscr{R}$ is a bisimulation relation, that $\langle\, \pi_n(y), \mu \,\rangle\checkmark$, and using the semantics of projection we finally find $\langle\, y, \mu \,\rangle\checkmark$.

(2) $\langle\, y, \mu \,\rangle\checkmark$. Similar to the previous case.

(3) $\langle\, x, \nu \,\rangle \xrightarrow{l} \langle\, x', \nu' \,\rangle$. We handle this case along the lines of [7]. Using the semantics of the projection operator, we find: $\langle\, \pi_{n+1}(x), \nu \,\rangle \xrightarrow{l} \langle\, \pi_n(x'), \nu' \,\rangle$, for any n. Furthermore, using the definition of $\mathscr{S}$, we find for every n that $(\pi_{n+1}(x), \nu)\mathscr{R}(\pi_{n+1}(y), \mu)$.
Now, we create a sequence $Q_n = \{(y', \mu') \mid \langle\, y, \mu \,\rangle \xrightarrow{l} \langle\, y', \mu' \,\rangle,\ (\pi_n(x'), \nu')\ \mathscr{R}(\pi_n(y'), \mu')\}$, and using the definition of projection and the fact that $\mathscr{R}$ is a bisimulation relation, we conclude that this sequence is non-empty for every n. Furthermore, it is decreasing ($Q_n \supseteq Q_{n+1}$) because in general we have $\pi_{n+1}(x) \leftrightarrows \pi_{n+1}(y) \Rightarrow \pi_n(x) \leftrightarrows \pi_n(y)$ and $\mathscr{R}$ contains all bisimulation relations that witness this. Lastly, every Q_n is finite, because y has bounded non-determinism. Therefore, the sequence Q_n eventually becomes constant. In other words, there exists (y'', μ'') such that for all n we have $(y'', \mu'') \in Q_n$. Hence, by definition of Q_n, we have for all n that

$(\pi_n(x'), \nu')\mathscr{R}(\pi_n(y''), \mu'')$. Now, using the definition of $\mathscr{S}$ and the fact that (y'', μ'') has bounded non-determinism because it is reachable from (y, μ), we finally conclude that $(x', \nu')\mathscr{S}(y'', \mu'')$.

(4) $\langle y, \mu \rangle \xrightarrow{l} \langle y', \mu' \rangle$. This case is also handled along the lines of [7]. Similarly to the previous case, we create a sequence $Q_n = \{(x', \nu') \mid \langle x, \nu \rangle \xrightarrow{l} \langle x', \nu' \rangle, (\pi_n(x'), \nu)\mathscr{R}(\pi_n(y'), \mu')\}$, and may conclude that this sequence is decreasing, and non-empty for every n. However, Q_n is not necessarily finite. Nevertheless, for every n and every $(x_n, \nu_n) \in Q_n$ there exists, using the previous case, a (y_n, μ_n) such that $\langle y, \mu \rangle \xrightarrow{l} \langle y_n, \mu_n \rangle$ and $(x_n, \nu_n)\mathscr{S}(y_n, \mu_n)$. Using bounded non-determinism of y, one of these elements occurs infinitely often. In other words, there is a k such that for every n there is an $m \geqslant n$ with $(y_k, \mu_k) \equiv (y_m, \mu_m)$. Now, because $x_k \mathscr{S} y_k$, we may conclude $\pi_n(x_k)\mathscr{R}\pi_n(y_k)$. Because $\mathscr{R}$ contains the identity relation, we find $\pi_n(y_k)\mathscr{R}\pi_n(y_m)$. Because $\mathscr{R}$ is symmetric, we find $\pi_n(y_m)\mathscr{R}\pi_n(x_m)$ and because $Q_m \subseteq Q_n$ we find $\pi_n(x_m)\mathscr{R}\pi_n(y')$. With transitivity of $\mathscr{R}$ we conclude $\pi_n(x_k)\mathscr{R}\pi_n(y')$ and finally $x_k \mathscr{S} y'$, which concludes the case.

In order to verify that $\mathscr{S}$ is robust, assume that $(x, \nu)\mathscr{S}(y, \mu)$. By definition of $\mathscr{S}$ we find that $(\pi_n(x), \nu)\mathscr{R}(\pi_n(y), \mu)$ for every n. Since $\mathscr{R}$ is robust, we may conclude for every interference ι and every n that $(\pi_n(x), \iota(\nu))\mathscr{R}(\pi_n(y), \iota(\mu))$, and hence $(x, \iota(\nu))\mathscr{S}(y, \iota(\mu))$. Therefore, $\mathscr{S}$ is also robust. □

Before we can use AIP^- to prove RSP, we need to study bounded non-determinism and projections of guarded recursive specifications in more detail. We need to show existence of a bounded non-deterministic solution for each guarded recursive specification, and we need an axiomatization for projection with respect to guarded process terms.

Theorem D.3 (Bounded non-determinism). *Each guarded recursive specification E has a bounded non-deterministic solution.*

Proof. This theorem is a strengthening of the recursive definition principle RDP, that states that every recursive specification has a solution. RDP is easily proven sound, using the fact that the semantics of HyPA actually gives one such solution. Let E be a guarded recursive specification. For the sake of convenience, assume that if $X \approx p \in E$, then p is already rewritten into a guarded process term, and furthermore assume that this term is of the form $\bigoplus_{j \in J} \left(d_j \gg a_j \odot q_j \oplus d'_j \gg c_j \rhd q'_j \oplus d''_j \gg \epsilon\right)$, where J is a finite set and q_j and q'_j are arbitrary process terms of HyPA. In this case, we can show that the solution defined by the semantics of HyPA, if we treat possibly occurring recursion variables as constants, has bounded non-determinism. Let $S \in \mathcal{V}_\mathrm{r} \to \mathcal{T}(\mathcal{V}_\mathrm{r})$ be the identity. I.e. the solution of E formed by the semantics of HyPA. By definition, every process, hence also every $S(X)$, with $X \in \mathcal{V}_\mathrm{r}$, has bounded non-determinism in 0 steps. If we then assume the induction hypothesis that every $S(X)$ has bounded non-determinism in n steps, we only need to prove that bounded non-determinism in $n + 1$ steps follows. By definition of the semantics of HyPA, we know for $S(X) = X \approx p$, that $\langle X, \nu \rangle \xrightarrow{l} \langle p', \nu \rangle$ if and only if $\langle p, \nu \rangle \xrightarrow{l} \langle p', \nu \rangle$. Using the specific form of p, and the semantics of HyPA, we know that there is only a finite number of these transitions, and that p' is either q_j or q'_j, for some j. Furthermore, the semantics of all process operators of HyPA is such that they lead to bounded non-deterministic compositions if the composed processes are bounded

non-deterministic. So, even if q_j and q'_j contain recursion variables, they are bounded non-deterministic in n steps, from which we may conclude that p, and hence $S(X)$, is bounded non-deterministic in $n+1$ steps. With induction, this concludes the proof, for the case where E is already rewritten as suggested above. For the case that the definitions in E still need to be rewritten, we may conclude only that there exists a bounded non-deterministic solution. It may not necessarily be the case that the solution defined by the semantics has this property. □

We claim, without proof, that the following axioms are sound for projection:

$\pi_n(\epsilon) \approx \epsilon$	$\pi_0(a) \approx \delta$	$\pi_0(c) \approx \delta$
$\pi_{n+1}(a \odot x) \approx a \odot \pi_n(x)$		$\pi_{n+1}(c \rhd x) \approx \pi_1(c) \rhd \pi_n(c \blacktriangleright x)$
$\pi_n(x \oplus y) \approx \pi_n(x) \oplus \pi_n(y)$		$\pi_n(d \gg x) \approx d \gg \pi_n(x)$
$\pi_n(x \odot y) \approx \pi_n(\pi_n(x) \odot y)$		$\pi_n(x \odot y) \approx \pi_n(x \odot \pi_n(y))$
$\pi_n(x \rhd y) \approx \pi_n(\pi_n(x) \rhd y)$		$\pi_n(x \rhd y) \approx \pi_n(x \rhd \pi_n(y))$
$\pi_n(x \between y) \approx \pi_n(\pi_n(x) \between y)$		$\pi_n(x \between y) \approx \pi_n(x \between \pi_n(y))$
$\pi_n(x \mid y) \approx \pi_n(x \mid \pi_n(y))$		$\pi_n(\pi_m(x)) \approx \pi_{\min(n,m)}(x)$

This brings us to the following two theorems.

Theorem D.4 (Projection push). *Define the interpretation* $\Pi_n \in \mathcal{V}_r \to \mathcal{T}(\mathcal{V}_r)$ *of recursion variables, such that* $\Pi_n(X) = \pi_n(X)$ *for all* $X \in \mathcal{V}_r$. *Then, for* $p \in \mathcal{T}(\mathcal{V}_r)$, *the following axiom is sound*:

$$\pi_n(p) \approx \pi_n(\Pi_n(p)),$$

where $\Pi_n(p)$ *denotes the application of* Π_n *to all the variables of* p.

Proof. It is straightforward from the axiomatization of projection, that any subterm p' of a process $\pi_n(p)$ may be replaced by $\pi_n(p')$, and if a subterm p' of $\pi_n(p)$ is of the form $\pi_n(p'')$, then it may be replaced by p''. □

Theorem D.5 (Guarded projection push). *Define the interpretation* $\Pi_n \in \mathcal{V}_r \to \mathcal{T}(\mathcal{V}_r)$ *as before, and let S be an arbitrary interpretation of recursion variables. Then, we find the following axioms for guarded process terms* p:

$$\pi_0(p) \approx \pi_0(S(p)), \quad \pi_{n+1}(p) \approx \pi_{n+1}(\Pi_n(p)).$$

Proof. Without loss of generality, assume that p is of the form

$$\bigoplus_{j\in J} \left(d_j \gg a_j \odot q_j \oplus d'_j \gg c_j \rhd q'_j \oplus d''_j \gg \epsilon\right),$$

with J finite, and q_j and q'_j arbitrary HyPA terms, possibly containing recursion variables. We use the axiomatization of projection to derive.

$$\begin{aligned}
\pi_0(p) &\approx \pi_0\left(\bigoplus_{j\in J} \left(d_j \gg a_j \odot q_j \oplus d'_j \gg c_j \rhd q'_j \oplus d''_j \gg \epsilon\right)\right)\\
&\approx \pi_0\left(\bigoplus_{j\in J} d''_j \gg \epsilon\right)\\
&\approx \pi_0\left(\bigoplus_{j\in J} \left(d_j \gg a_j \odot S(q_j) \oplus d'_j \gg c_j \rhd S(q'_j) \oplus d''_j \gg \epsilon\right)\right)\\
&\approx \pi_0(S(p)).
\end{aligned}$$

More elaborately, we also use the projection push to find:

$$\begin{aligned}
&\pi_{n+1}(p)\\
&\approx \pi_{n+1}\left(\textstyle\bigoplus_{j\in J}\,(d_j \gg a_j \odot q_j \oplus d'_j \gg c_j \rhd q'_j \oplus d''_j \gg \epsilon)\right)\\
&\approx \pi_{n+1}\left(\textstyle\bigoplus_{j\in J}\,(d_j \gg a_j \odot \pi_n(q_j) \oplus d'_j \gg c_j \rhd \pi_n(q'_j) \oplus d''_j \gg \epsilon)\right)\\
&\approx \pi_{n+1}\left(\textstyle\bigoplus_{j\in J}\,(d_j \gg a_j \odot \pi_n(\Pi_n(q_j)) \oplus d'_j \gg c_j \rhd \pi_n(\Pi_n(q'_j)) \oplus d''_j \gg \epsilon)\right)\\
&\approx \pi_{n+1}\left(\textstyle\bigoplus_{j\in J}\,(d_j \gg a_j \odot \Pi_n(q_j) \oplus d'_j \gg c_j \rhd \Pi_n(q'_j) \oplus d''_j \gg \epsilon)\right)\\
&\approx \pi_{n+1}\left(\Pi_n(p)\right).
\end{aligned}$$

This concludes the proof. □

Now, using the guarded projection push theorem, the theorem on bounded non-determinism of guarded recursive specifications, and AIP^-, it is easy to derive soundness of RSP.

Theorem D.6. *The recursive specification principle is sound.*

Proof. For convenience assume that $X \approx p \in E$ implies that p is already rewritten into a guarded term. Using the theorem on bounded non-determinism, we know that there exists a solution S of E that has bounded non-determinism, i.e. $B(S(X))$ for every $X \in \mathcal{V}_{\mathrm{r}}$. Suppose that S' is an arbitrary other solution for E. We will show by induction on n that for every $X \in \mathcal{V}_{\mathrm{r}}$ we have $\pi_n(S(X)) \approx \pi_n(S'(X))$. From that we then may conclude $S(X) \approx S'(X)$ using AIP^-. Note, that if we have two arbitrary solutions of E, that we may conclude them equal by showing that both are equal to S.

The base case, where $n = 0$, is derived using congruence (derivation rule (4)) and the first part of the guarded projection theorem:

$$\pi_0(S(X)) \approx \pi_0(S(p)) \approx \pi_0(p) \approx \pi_0(S'(p)) \approx \pi_0(S'(X)).$$

Using the second part of the guarded projection theorem, and the induction hypothesis that $\pi_n(S(X)) \approx \pi_n(S'(X))$ we find firstly, using congruence again, that $S(\Pi_n(p)) \approx S'(\Pi_n(p))$, and using this we derive:

$$\begin{aligned}
\pi_{n+1}(S(X)) &\approx \pi_{n+1}(S(p)) \approx S(\pi_{n+1}(p)) \approx S(\pi_{n+1}(\Pi_n(p)))\\
&\approx \pi_{n+1}(S(\Pi_n(p))) \approx \pi_{n+1}(S'(\Pi_n(p))) \approx S'(\pi_{n+1}(\Pi_n(p)))\\
&\approx S'(\pi_{n+1}(p)) \approx \pi_{n+1}(S'(p)) \approx \pi_{n+1}(S'(X)). \qquad \square
\end{aligned}$$

Appendix E. Rewriting into basic terms

In this appendix, we prove that every closed HyPA term is derivably equal to a basic term. Thereto, let p be an arbitrary closed HyPA term. For the first step of this proof, assume that all occurrences of δ, ϵ, atomic actions a and flow clauses c are underlined. We use the notation $\underline{p}$ for the underlined version of p. On this underlined version of p we apply the rewrite system consisting of the following four rewrite rules:

$$\begin{aligned}
\underline{\delta} &\hookrightarrow \big[\mathit{false}\big] \gg \epsilon,\\
\underline{\epsilon} &\hookrightarrow \big[\mathit{true}\big] \gg \epsilon,\\
\underline{a} &\hookrightarrow \big[\mathit{true}\big] \gg a \odot \big[\mathit{true}\big] \gg \epsilon,\\
\underline{c} &\hookrightarrow \big[\mathit{true}\big] \gg c \rhd \big[\mathit{false}\big] \gg \epsilon.
\end{aligned}$$

First, observe that this term rewrite system is strongly normalizing as in each rewrite step the number of underlined symbols decreases. Second, all four rewrite rules are derivable using the axioms of HyPA (neglecting the underlining):

$$\delta \approx \underline{[false]} \gg \epsilon,$$
$$\epsilon \approx \underline{[true]} \gg \epsilon,$$
$$a \approx a \odot \epsilon \approx \underline{[true]} \gg a \odot \epsilon \approx \underline{[true]} \gg a \odot \underline{[true]} \gg \epsilon,$$
$$c \approx c \rhd \delta \approx \underline{[true]} \gg c \rhd \delta \approx \underline{[true]} \gg c \rhd \underline{[false]} \gg \epsilon.$$

Finally, the normal forms of underlined versions of closed HyPA terms are necessarily of the form

$$\begin{aligned} N' ::= {} & d \gg \epsilon \mid d \gg a \odot N' \mid d \gg c \rhd N' \mid N' \oplus N' \\ & \mid d \gg N' \mid N' \odot N' \mid N' \blacktriangleright N' \mid N' \rhd N' \\ & \mid N' \parallel N' \mid N' \between N' \mid N' \mid N' \mid \partial_H(N'). \end{aligned}$$

Observe that basic terms are also of this form. Thus we have achieved that for any closed HyPA term there exists an N' term that is derivably equal. In the remainder of this appendix, we show that for any N' term there exists a basic term that is derivably equal.

E.1. The rewrite system

Next, we give a rewrite system that is constructed for the task of rewriting N' terms into basic terms.

(1) $d \gg d' \gg x \to (d \sim d') \gg x$
(2) $d \gg (x \oplus y) \to d \gg x \oplus d \gg y$

(3) $(d \gg \epsilon) \odot x \to d^{?} \gg x$
(4) $(d \gg a \odot x) \odot y \to d \gg a \odot (x \odot y)$
(5) $(d \gg c \rhd x) \odot y \to d \gg c \rhd x \odot y$
(6) $(x \oplus y) \odot z \to x \odot z \oplus y \odot z$

(7) $x \blacktriangleright y \to x \rhd y \oplus y$
(8) $(d \gg \epsilon) \rhd x \to d \gg \epsilon$
(9) $(d \gg a \odot x) \rhd y \to d \gg a \odot (x \blacktriangleright y)$
(10) $(d \gg c \rhd x) \rhd y \to d \gg c \rhd (x \blacktriangleright y)$
(11) $(x \oplus y) \rhd z \to x \rhd z \oplus y \rhd z$

(12) $\partial_H(d \gg \epsilon) \to d \gg \epsilon$
(13) $\partial_H(d \gg a \odot x) \to d \gg a \odot \partial_H(x)$ if $a \notin H$
(14) $\partial_H(d \gg a \odot x) \to \underline{[false]} \gg \epsilon$ if $a \in H$
(15) $\partial_H(d \gg c \rhd x) \to d \gg c \rhd \partial_H(x)$
(16) $\partial_H(x \oplus y) \to \partial_H(x) \oplus \partial_H(y)$

(17) $x \parallel y \to x \between y \oplus y \between x \oplus x \mid y$
(18) $d \gg \epsilon \between x \to \underline{[false]} \gg \epsilon$
(19) $d \gg a \odot x \between y \to d \gg a \odot (x \parallel y)$
(20) $d \gg c \rhd x \between y \to \underline{[false]} \gg \epsilon$
(21) $(x \oplus y) \between z \to x \between z \oplus y \between z$

$$
\begin{array}{lll}
(22)\ (x \oplus y) \mid z & \hookrightarrow & x \mid z \oplus y \mid z \\
(23)\ x \mid (y \oplus z) & \hookrightarrow & x \mid y \oplus x \mid z \\
(24)\ d \gg \epsilon \mid d' \gg \epsilon & \hookrightarrow & (d^? \wedge d'^?) \gg \epsilon \\
(25)\ d \gg \epsilon \mid d' \gg a \odot x & \hookrightarrow & \lfloor false \rfloor \gg \epsilon \\
(26)\ d \gg a \odot x \mid d' \gg \epsilon & \hookrightarrow & \lfloor false \rfloor \gg \epsilon \\
(27)\ d \gg \epsilon \mid d' \gg c \rhd x & \hookrightarrow & (d^? \sim d') \gg c \rhd x \\
(28)\ d \gg c \rhd x \mid d' \gg \epsilon & \hookrightarrow & (d'^? \sim d) \gg c \rhd x \\
(29)\ d \gg a \odot x \mid d' \gg a' \odot y & \hookrightarrow & (d \wedge d') \gg (a \gamma a') \odot (x \parallel y) \text{ if } (a \gamma a') \text{ defined} \\
(30)\ d \gg a \odot x \mid d' \gg a' \odot y & \hookrightarrow & \lfloor false \rfloor \gg \epsilon \text{ if } (a \gamma a') \text{ undefined} \\
(31)\ d \gg a \odot x \mid d' \gg c \rhd y & \hookrightarrow & \lfloor false \rfloor \gg \epsilon \\
(32)\ d \gg c \rhd x \mid d' \gg a \odot y & \hookrightarrow & \lfloor false \rfloor \gg \epsilon \\
(33)\ d \gg c \rhd x \mid d' \gg c' \rhd y & \hookrightarrow & ((d \sim c_{jmp}) \wedge (d' \sim c'_{jmp})) \gg (c \wedge c') \rhd \\
 & & \begin{pmatrix} x \between (\lfloor true \rfloor \gg c' \rhd \lfloor false \rfloor \gg \epsilon) \blacktriangleright y \oplus \\ y \between (\lfloor true \rfloor \gg c \rhd \lfloor false \rfloor \gg \epsilon) \blacktriangleright x \oplus \\ x \mid (\lfloor true \rfloor \gg c' \rhd \lfloor false \rfloor \gg \epsilon) \blacktriangleright y \oplus \\ y \mid (\lfloor true \rfloor \gg c \rhd \lfloor false \rfloor \gg \epsilon) \blacktriangleright x \end{pmatrix}
\end{array}
$$

In the following section we show that this rewrite system only allows to rewrite N' terms into derivably equal N' terms (soundness of the rewrite system, see Appendix E.2), that the rewrite system is strongly normalizing (see Appendix E.3), and that every normal form of an N' term is necessarily a basic term (see Appendix E.4).

E.2. Soundness of the rewrite system

In this subsection we show that for each rewrite rule $s \hookrightarrow t$ of the rewrite system introduced in Appendix E.1, we have HyPA $\vdash s = t$.

For the rewrite rules (1)–(3), (6), (7), (11), (16), (17), (21)–(24), (27), and (29), this follows directly from the axioms as for each of these rewrite rules there is an axiom that states that the left-hand and right-hand sides are derivably equal. For the rewrite rules (25), (30), and (32) this is obtained from the axioms and application of $\delta \approx \lfloor false \rfloor \gg \epsilon$ and/or $c \approx \lfloor true \rfloor \gg c \rhd \lfloor false \rfloor \gg \epsilon$. Both these equalities have been proven in first step of the elimination result in this appendix. For the rewrite rules (26), (28), (31), and (33) this follows from the soundness of other rewrite rules and the axiom $x \mid y \approx y \mid x$. For the other rewrite rules the derivations are shown below:

$$
\begin{array}{ll}
(4) & (d \gg a \odot x) \odot y \approx ((d \gg a) \odot x) \odot y \approx (d \gg a) \odot (x \odot y) \\
 & \quad \approx d \gg a \odot (x \odot y) \\
(5) & (d \gg c \rhd x) \odot y \approx ((d \gg c) \rhd x) \odot y \approx ((d \gg c) \odot \delta \rhd x) \odot y \\
 & \quad \approx (d \gg c) \odot \delta \rhd x \odot y \approx (d \gg c) \rhd x \odot y \approx d \gg c \rhd x \odot y \\
(8) & (d \gg \epsilon) \rhd x \approx d \gg \epsilon \rhd x \approx d \gg \epsilon \\
(9) & (d \gg a \odot x) \rhd y \approx d \gg a \odot x \rhd y \approx d \gg a \odot (x \blacktriangleright y) \\
(10) & (d \gg c \rhd x) \rhd y \approx d \gg (c \rhd x) \rhd y \approx d \gg c \rhd (x \blacktriangleright y) \\
(12) & \partial_H (d \gg \epsilon) \approx d \gg \partial_H (\epsilon) \approx d \gg \epsilon \\
(13,14) & \partial_H (d \gg a \odot x) \approx d \gg \partial_H (a \odot x) \approx d \gg \partial_H (a) \odot \partial_H (x) \\
 & \quad \approx \begin{cases} d \gg a \odot \partial_H (x) & \text{if a} \notin H \\ d \gg \delta \odot \partial_H (x) \approx d \gg \delta \approx \delta \approx \lfloor false \rfloor \gg \epsilon & \text{if a} \in H \end{cases}
\end{array}
$$

(15) $\partial_H(d \gg c \rhd x) \approx d \gg \partial_H(c \rhd x) \approx d \gg c \rhd \partial_H(x)$
(18) $d \gg \epsilon \between x \approx d \gg (\epsilon \between x) \approx d \gg \delta \approx \delta \approx [\mathit{false}] \gg \epsilon$
(19) $d \gg a \odot x \between y \approx d \gg (a \odot x \between y) \approx d \gg a \odot (x \parallel y)$
(20) $d \gg c \rhd x \between y \approx d \gg (c \rhd x \between y) \approx d \gg \delta \approx \delta \approx [\mathit{false}] \gg \epsilon$

E.3. The rewrite system is strongly normalizing

That the above rewrite system is strongly normalizing can be demonstrated using semantical labeling in combination with the recursive path ordering technique as (among others) described in [41]. We define the following ranking-norm on N' terms:

- $\lfloor \epsilon \rfloor = \lfloor a \rfloor = 0$;
- $\lfloor c \rfloor = 1$;
- $\lfloor d \gg x \rfloor = \lfloor \partial_H(x) \rfloor = \lfloor x \rfloor + 1$;
- $\lfloor x \oplus y \rfloor = \lfloor x \odot y \rfloor = \lfloor x \blacktriangleright y \rfloor = \lfloor x \rhd y \rfloor = \lfloor x \parallel y \rfloor = \lfloor x \between y \rfloor = \lfloor x \mid y \rfloor = \lfloor x \rfloor + \lfloor y \rfloor$.

Now, we label the operators $d \gg \_$, $\odot$, $\blacktriangleright$, $\rhd$, $\parallel$, $\between$, and $\mid$ with the norm of the term they are the leading symbols of. I.e. we write $x \odot_{\lfloor x \rfloor + \lfloor y \rfloor} y$ in stead of $x \odot y$. Then, we define the following (well-founded) ordering on labeled operators. (Note that we still treat $d \gg x$ as a unary operator.)

- ϵ, a (for $a \in A$) and $\oplus$ are smaller than all other operators;
- $d \gg_n < d' \gg_{n+1}$ for all n, d, d';
- $d \gg_n \_ < \odot_0$ for all n;
- $\odot_n < \rhd_n < \blacktriangleright_n < \odot_{n+1}$ for all n;
- $\epsilon < \partial_H()$;
- $\blacktriangleright_n < \partial_H()$ for all n;
- $\blacktriangleright_n < \between_0$ for all n;
- $\between_n < \mid_n < \parallel_n < \between_{n+1}$ for all n.

It is straightforward, but cumbersome, to show for each of the rules that they are strictly decreasing with respect to the recursive path ordering based on $<$.

E.4. Normal forms are basic terms

Now, we prove that every normal form of an N' term is a basic term. Let s be such a normal form. Suppose that s is not a basic term. Then, as the term rewrite system only rewrites N' terms into N' terms and all basic terms are N' terms, s must contain a smallest subterm s' which is not a basic term. Then, s' is of one of the following forms: $d \gg s_1$, $s_1 \odot s_2$, $s_1 \blacktriangleright s_2$, $s_1 \rhd s_2$, $s_1 \parallel s_2$, $s_1 \between s_2$, $s_1 \mid s_2$, or $\partial_H(s_1)$ for some N' terms s_1 and s_2. Actually, as s' is a smallest subterm of s that is not a basic term, necessarily, s_1 and s_2 are basic terms. By inspection of the rewrite rules it is obvious that in any case a rewrite rule is applicable. Hence, s' is not a normal form. This contradiction leads to the conclusion that s' is a basic term. Hence, every normal form of an N' term is a basic term.

References

[1] P. Mosterman, Hybrid dynamic systems: a hybrid bond graph modeling paradigm and its application in diagnosis, Ph.D. thesis, Vanderbilt University, Nashville, Tennessee, 1997.

[2] A. van der Schaft, J. Schumacher, An Introduction to Hybrid Dynamical Systems, in: Lecture Notes in Control and Information Sciences, vol. 251, Springer-Verlag, London, 2000.
[3] J. Groote, M. Reniers, Algebraic process verification, in: J. Bergstra, A. Ponse, S. Smolka (Eds.), Handbook of Process Algebra, Elsevier Science, Amsterdam, 2001, pp. 1151–1208 (Chapter 17).
[4] V. Bos, J. Kleijn, Formal specification and analysis of industrial systems, Ph.D. thesis, TU/e, Eindhoven, The Netherlands, 2002.
[5] V. Bos, J. Kleijn, Redesign of a systems engineering language—formalisation of χ, Formal Aspects of Computing 15 (4) (2003) 370–389.
[6] W. Fokkink, J. Groote, M. Hollenberg, B. van Vlijmen, LARIS 1.0: LAnguage for Railway Interlocking Specifications, CWI, Amsterdam, 2000.
[7] J. Baeten, W. Weijland, Process Algebra, in: Cambridge Trancts in Theoretical Computer Science, vol. 18, Cambridge University Press, Cambridge, 1990.
[8] E. Brinksma, A tutorial on LOTOS, in: M. Diaz (Ed.), Proc. Protocol Specification, Testing and Verification V, Amsterdam, The Netherlands, 1985, pp. 171–194.
[9] R. Schiffelers, D. van Beek, K. Man, M. Reniers, J. Rooda, A hybrid language for modeling, simulation and verification, in: Proc. IFAC Conference on Analysis and Design of Hybrid Systems (ADHS03), International Federation of Automatic Control (IFAC), Saint-Malo, 2003, pp. 235–240.
[10] D. van Beek, N. Jansen, K. Man, M. Reniers, J. Rooda, R. Schiffelers, Relating Chi to hybrid automata, in: S.Chick, P. Sánchez, D. Ferrin, D. Morrice (Eds.), Proceedings Winter Simulation Conference (WSC03), New Orleans, USA, 2003, pp. 632–640.
[11] R. Schiffelers, D. van Beek, K. Man, M. Reniers, J. Rooda, Formal semantics of hybrid Chi, in: Formal Modelling and Analysis of Timed Systems, Lecture Notes in Computer Science, Springer-Verlag, in press.
[12] J. Vereijken, A process algebra for hybrid systems, in: The Second European Workshop on Real-Time and Hybrid Systems, Grenoble, France, 1995.
[13] J. Bergstra, C. Middelburg, Process algebra for hybrid systems, Tech. Rep. CSR 03-06, TU/e, Eindhoven, The Netherlands, 2003.
[14] H. Jifeng, From CSP to hybrid systems, in: A.W. Roscoe (Ed.), A Classical Mind, Essays in Honour of C.A.R. Hoare, Prentice-Hall International, 1994, pp. 171–189.
[15] W. Rounds, H. Song, The ϕ-calculus: a language for distributed control of reconfigurable embedded systems, in: F. Wiedijk, O. Maler, A. Pnueli (Eds.), Hybrid Systems: Computation and Control, 6th International Workshop, HSCC 2003, Lecture Notes in Computer Science, vol. 2623, Springer-Verlag, 2003, pp. 435–449.
[16] T. Henzinger, The theory of hybrid automata, in: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996), IEEE Computer Society Press, 1996, pp. 278–292.
[17] N. Lynch, R. Segala, F. Vaandrager, Hybrid I/O automata, Information and Computation 185 (1) (2003) 105–157.
[18] J. LeBail, H. Alla, R. David, Hybrid Petri nets, in: Proc. of the 1st European Control Conference, ECC'91, Grenoble, France, July, 1991, pp. 1472–1477.
[19] H. Alla, R. David, Continuous and hybrid Petri nets, Journal of Circuits, Systems and Computers 8 (1) (1998) 159–188.
[20] I. Demongodin, N. Koussoulas, Differential Petri nets: A new model for hybrid systems, in: Proc. Advanced Summer Institute '96, 1996, pp. 61–68.
[21] I. Demongodin, N. Koussoulas, Differential Petri nets: Representing continuous systems in a discrete-event world, IEEE Transactions on Automatic Control 43 (3) (1998) 573–579.
[22] A. Febbraro, A. Giua, G. Menga (Eds.), Special Issue on Hybrid Petri Nets, in: Discrete Event Dynamic Systems, vol. 11, 2001.
[23] M. Rönkkö, A. Ravn, K. Sere, Hybrid action systems, Theoretical Computer Science 290 (2003) 937–973.
[24] P. Amthor, A CSP model for hybrid automata, in: Third BCS-FACS Northern Formal Methods Workshop (NFMW98), Ilkley, UK, 1998.
[25] T. Willemse, Semantics and verification in process algebras with data and timing, Ph.D. thesis, TU/e, Eindhoven, The Netherlands, 2003.
[26] J. Groote, J. van Wamel, Analysis of three hybrid systems in timed μCRL, Science of Computer Programming 39 (2001) 215–247.
[27] A. van der Schaft, J. Schumacher, Compositionality issues in discrete, continuous, and hybrid systems, Int. J. Robust and Nonlinear Control 11 (2001) 417–434.
[28] A. Bemporad, M. Morari, Control of systems integrating logic, dynamics, and constraints, Automatica 35 (3) (1999) 407–427.

[29] W. Heemels, B. de Schutter, A. Bemporad, On the equivalence of classes of hybrid dynamical models, in: Proc. 40th IEEE Conference on Decision and Control, IEEE, Orlando, FL, 2001, pp. 364–369.
[30] A. van der Schaft, J. Schumacher, Complementarity modeling of hybrid systems, IEEE Transactions on Automatic Control 43 (1998) 483–490.
[31] E.D. Sontag, Nonlinear regulation: The piecewise linear approach, IEEE Trans. Autom. Control 26 (1981) 346–358.
[32] E. Haghverdi, P. Tabuada, G. Pappas, Bisimulation relations for dynamical and control systems, in: R. Blute, P. Selinger (Eds.), Category Theory and Computer Science (CTCS'02), Electronic Notes in Theoretical Computer Science, vol. 69, Elsevier, 2003.
[33] G. Lafferriere, G. Pappas, S. Sastry, O-minimal hybrid systems, Mathematics of Control, Signals, and Systems 13 (1) (2000) 1–21.
[34] P.-H. Ho, Automatic analysis of hybrid systems, Ph.D. thesis, Cornell University, Ithaca, New York, 1995.
[35] R. Alur, T. Henzinger, P.-H. Ho, Automatic symbolic verification of embedded systems, IEEE Transactions on Software Engineering 22 (3) (1996) 181–201.
[36] J. Polderman, J. Willems, Introduction to Mathematical Systems Theory: A Behavioural Approach, in: Texts in Applied Mathematics, vol. 26, Springer-Verlag, 1998.
[37] R. Dorf, R. Bishop, Modern Control Systems, Series in Electrical and Computer Engineering: Control Engineering, Addison-Wesley, 1995.
[38] P. Cuijpers, M. Reniers, W. Heemels, Hybrid transition systems, Tech. Rep. CSR 02-12, TU/e, Eindhoven, The Netherlands, 2002.
[39] E. Sontag, Mathematical Control Theory: Deterministic Finite Dimensional Systems, in: Texts in Applied Mathematics, vol. 6, Springer-Verlag, 1998.
[40] G. Plotkin, A structural approach to operational semantics, Tech. Rep. DAIMI FN-19, Computer Science Department, Aarhus University, 1981.
[41] J. Baeten, C. Verhoef, Concrete process algebra, in: S. Abramsky, D.M. Gabbay, T. Maibaum (Eds.), Semantic Modelling, in: Handbook of Logic in Computer Science, vol. 4, 1995, pp. 149–268.
[42] R. Milner, A Calculus of Communicating Systems, in: Lecture Notes in Computer Science, vol. 92, Springer-Verlag, 1980.
[43] W. Fokkink, Introduction to Process Algebra, Texts in Theoretical Computer Science, Springer-Verlag, Berlin, 1998.
[44] J. Baeten, J. Bergstra, Mode transfer in process algebra, Tech. Rep. CSR 00-01, TU/e, Eindhoven, The Netherlands, 2000.
[45] J. Baeten, C. Middelburg, Process Algebra with Timing, Monographs in Theoretical Computer Science, Springer-Verlag, 2002.
[46] P. Cuijpers, M. Reniers, Hybrid process algebra, Tech. Rep. CSR 03-07, TU/e, Eindhoven, The Netherlands, 2003.
[47] J.-R. Abrial, Steam-boiler control specification problem, in: Dagstuhl Meeting: Methods for Semantics and Specification, 1995.
[48] J. Baeten, T. Basten, M. Reniers, Algebra of communicating processes, course notes 2M920: Process Algebra, 2003.
[49] J. Bergstra, J. Klop, Verification of an alternating bit protocol by means of process algebra, in: W.B.K. Jantke (Ed.), Mathematical Methods of Specification and Synthesis of Software Systems '85, Lecture Notes in Computer Science, vol. 215, Springer, 1986, pp. 9–23.
[50] S. Owicki, D. Gries, An axiomatic proof technique for parallel programs I, Acta Informatica 6 (1976) 319–340.
[51] J. Rooda, Simulation of Logistics Elements (Sole), Enschede, The Netherlands, user Manual, 1982.
[52] J. Baeten, J. Bergstra, Process algebra with propositional signals, Theoretical Computer Science 177 (1997) 381–405.
[53] P. Cuijpers, M. Reniers, Topological (bi-)simulation, Tech. Rep. CSR 02-04, TU/e, Eindhoven, The Netherlands, 2002.
[54] M. Ying, Topology in Process Calculus: Approximate Correctness and Infinite Evolution of Concurrent Programs, Springer-Verlag, 2001.
[55] R. van Glabbeek, The linear time—branching time spectrum I: The semantics of concrete, sequential processes, in: J. Bergstra, A. Ponse, S. Smolka (Eds.), Handbook of Process Algebra, Elsevier Science, Amsterdam, 2001, pp. 3–99 (Chapter 1).